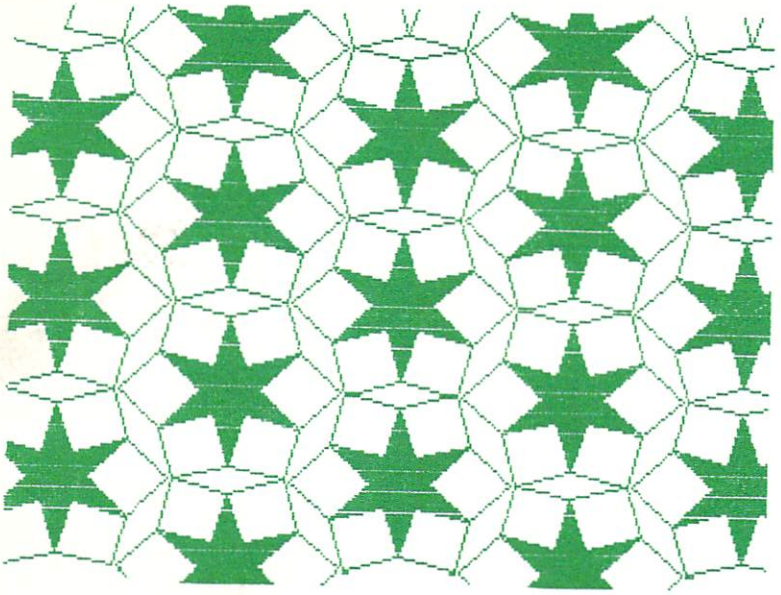
IBM PC COMAL 2.0 ANNOUNCED BY IBM DENMARK



CELEBRATING COMALS 10TH ANNIVERSARY

COMAL 5 TODAY

$3.95

I SPEAK

COMAL

EXPAND COMAL 0.14
TO 11,838 BYTES FREE

COMAL IMPLEMENTATIONS

JUST FOR BEGINNERS

OKIDATA 92 SUPPORT
PLUS CHINESE SCREENS

COMAL 0.14 SET OF
MACHINE CODE ROUTINES

SHOW STOPPER - CAT PAUSER

SIGN LANGUAGE

SPANISH COMAL

COMAL BULLETIN BOARDS

PROGRAMMERS ALARM SYSTEM

DISK EDITOR

POLAR GRAPHS WITH COMAL

COMAL 2.0 LIGHT PEN USE

PROGRAMS ON 28 COMAL DISKS
COMPLETE DIRECTORIES LISTED
OVER 1,000 COMAL PROGRAMS

TABLE OF CONTENTS - COMAL TODAY #5

1 - How To Type In Programs
  - Sieve Benchmark
  - Compute! Benchmark
2 - COMAL 2.0 Cartridge Story
3 - Spanish COMAL
4 - Cartridge Demo Disk 3 Story
5 - Memory Chart for COMAL 2.0
6 - COMAL Implementations
    by Kevin Ryan
7 - Missed IBM PC Sale
  - Ignored Again
8 - Just For Beginners
10- COMAL 2.0 Input Field
11- Over 1,000 COMAL Programs
  - Eight Queens
12- The Missing Comma
  - How COMAL Statements Are Stored
    by John H McCoy
15- Letter From A Cartridge User
16- Best Selling COMAL Books
  - COMAL 0.14 Keyword Table, Debug, &
17- COMAL Can Compete With BASIC
    by Colin Thompson
18- Okidata 92 Printer Support
    by Craig Van Degrift
  - Chinese Screens
    by Craig Van Degrift
19- Expand COMAL 0.14 Memory to
    11,838 Bytes Free
    by John H McCoy
20- Fun With COMAL 2.0
  - 127 COMAL Colors
21- Attack Of The Mutant COMAL's
    by David Skinner
22- COMAL 0.14 - Just For Fun
23- Strings In COMAL
    by Len Lindsay
24- You're Looking Good - or -
    Unlearning Some BASIC Ideas
    by Steve Kortendick
25- Interrupt Input - Easy Restart
    by William Wood
26- COMAL 0.14 Machine Code Routines
    by David Stidolph & Jesse Knight
30- COMAL 2.0 Modem Use
    by David Stidolph
31- Target Game - A Start
    by Len Lindsay
  - Show Stopper - CAT Pauser
    by Ray Carter
32- Dog Chases Cat
    by Bert Denaci
33- Sign Language
    by Carl Frerichs
  - Show Your Own Turtle
    by David Stidolph
34- How To Help COMAL
35- Cartridge And IEEE Interfaces
  - COMAL Bulletin Boards
36- Printer Notes / Select Disk File
    by Len Lindsay
  - Bank Switching & Virtual Memory
37- PROMAL Review
  - Savescreen Correction
38- Programmers Alarm System
    by Ray Carter & Jerry Claessen
39- 1541 Flash Works With COMAL 2.0
  - Testing Logic Circuits
    by Tom Kuiper
40- Professional Graphs
    by Tom Kuiper
42- Signals From Sid
    by Tom Kuiper
43- STR$ In Any Number Base
    by Tom Kuiper
44- Pitfall Harry - COMAL Style
45- Disk Editor
    by Ian MacPhedran
46- See Program Name Table
    by David Stidolph
47- Clear The Keyboard Buffer
    by Len Lindsay
  - Inventory Program
    by Gordon Shigley
49- Tutorial Disk Strikes Back
50- Polar Graphs With COMAL
    by Ron France
51- 1520 Plotter 2.0 Graphics Dump
52- Guess It
    by Carl Frerichs
54- COMAL Language Notes
    by Carl Frerichs
55- COMAL 2.0 Light Pen Use
    by Bob Schulz
57- Disk Directories Listing
64- COMAL User Groups
  - NEC Double Size Graphics Dump
    by Peter Foiles

ADVERTISERS INDEX:

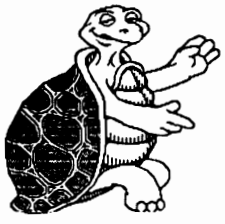
51- Northwest Users Guide
BC- United States Commodore User Group
  - Transactor Magazine

COMAL TODAY SUBSCRIBER SPECIALS:

See the inside back cover

COMAL RESOURCE LIST:

See the back cover



Hi, I'm Calvin the COMAL Turtle™ I'll be back next issue with some neat Turtle Graphics just for you. See you then.

COMAL TODAY ISSUE #5, December 21, 1984
Editor: Len Lindsay
Publisher: COMAL Users Group, U.S.A., Ltd.
5501 Groveland Ter, Madison, WI 53716
Calvin the COMAL Turtle by: Wayne Schmidt

HOW TO TYPE IN PROGRAMS

Line numbers are irrelevant to a running COMAL program. COMAL only provides line numbers for our benefit in editing the program. Thus most magazines do not use line numbers when listing a COMAL PROGRAM. It is up to **YOU** to provide the line numbers. But of course, **COMAL can do it for you** quite easily. Just follow these steps to type in a COMAL program listing:

1) Enter command: NEW
2) Enter command: AUTO
3) Type in the program
4) Hit RETURN key twice when done

Remember - use unshifted letters thoughout entering the program. If letters are capitalized in the listing it does not mean use the SHIFT with those letters. They are capitalized merely for a pleasant, easy to read, look. The only place to use SHIFTED letters is inside quotes. Also, you don't have to type leading spaces in a line. They are listed only to emphasize structures.

LONG PROGRAM LINES: We are continuing to print COMAL TODAY with two columns per page, printed in elite (12 pitch) with 40 characters maximum per line. This makes it easiest to read. However, some program listings have program lines that extend beyond the 40 character limit. We decided to list these lines in the same manner that COMAL uses when listing long lines on a 40 column screen. We simply break the line right at the 40 character spot, and continue it on the next line, indenting it properly to keep the program structures obvious. These are called wrap lines. To draw your attention to these continued lines we add a **//wrap line** comment to the end of the line. Whenever you see this make sure you type both lines as one continuous program line! The following example includes a line with more than 40 characters that we must list on two lines, but you must type in as one long program line:

```
IF CURRENT'NAME$<>"STOP" THEN PRINT'LABE
L(CURRENT'NAME$,PHONE$) //wrap line
```

If you type in this long program line as two shorter program lines, COMAL will not object (although sometimes it will)! But, the program will not work unless it is entered as one long line. The procedure name PRINT'LABEL is split onto two lines in the listing, but the //wrap line draws your attention to this fact.

SIEVE BENCHMARK

The October issue of SPRITE 64 (Michigan Commodore-64 Users Group, Inc.) ran a benchmark with Comal, Basic, Simons Basic and Oxford Pascal. We ran the benchmark in COMAL 2.0. Oxford Pascal ran only slightly faster than COMAL 2.0, and it is compiled! Here are the results:

Oxford Pascal	923 Jiffies
COMAL 2.0	1046 Jiffies
COMAL 0.14	2358 Jiffies
C64 BASIC	3326 Jiffies
Simons Basic	3517 Jiffies

COMPUTE! BENCHMARK

The January issue of COMPUTE! ran a benchmark they claimed was revealing since it tested typical operations. The IBM PC was the fastest, but no match for COMAL. Here are some of the results:

COMAL 2.0	3 Min 34 Sec
IBM PC	5 Min 45 Sec
GOLDSTAR MSX	6 Min 20 Sec
IBM PCjr	6 Min 59 Sec
COMMODORE 64	7 Min 2 Sec
COMMODORE PLUS 4	8 Min 36 Sec

COMAL 2.0 CARTRIDGE STORY

A REPORT BY CAPTAIN COMAL

We could write a book about the SAGA of the COMAL Cartridge - possibly the most significant product developed for the C64. It seemed every week there were changes in developement, production, manufacture, and distribution plans. Now, things are finally calming down and distribution is under way, nearly 1,000 cartridges have been shipped.

In the summer of 1984 Commodore told us that they were making the COMAL 2.0 Cartridges. We were told that ROM production had started, and mass production would begin soon. We were told to expect delivery in October 1984. So we announced November delivery here and placed an order for HALF of the first production run. As a thank you for early support, we decided to give all who ordered the Cartridge in ADVANCE of it's first shipping date, a free set of 2 books and 2 disks.

Then problems arose. Commodore cancelled ROM production and halted plans for making the cartridge. What to do? We decided to make the cartridges ourselves! We promised the cartridge for November 1984. We wanted to keep that promise. We ordered 4,000 EPROMs (27128 type) to make 1,000 cartridges. Expensive! Then we had to get EPROM burning equipment and arrange for people to program each EPROM. Then we ordered the PC boards and plastic cases. We had to put each EPROM chip in the sockets on each PC board, one at a time. Lots of work. And our total cost was more than the price we announced. Yes, we were going to do all this extra work just to get the cartridge out - and then lose money doing it. We think it was worth it. (Plus since all our efforts went to making the cartridge this issue of COMAL TODAY was delayed).

We made a special trip to Denmark October 29 thru November 2, 1984 to sign formal agreements with UniComal Aps - authors and owners of the C64 COMAL Cartridge. We continue to be fully licensed to make and distribute the Cartridge in North America. We pay royalties to UniComal.

On October 31, 1984 Commodore Denmark finally signed a contract with UniComal

Aps for rights to manufacture and distribute the cartridge worldwide. This agreement does not take away or restrict our rights for North American Distribution of the cartridge ourselves. Apparantly Commodore had no cartridge contract prior to this. One provision in their contract guarantees North American distribution rights to COMAL Users Group, U.S.A., Limited.

Commodore USA continues to have no interest in distributing the cartridge according to our latest information. If you want a cartridge you get it from COMAL Users Group, NOT Commodore.

The COMAL Cartridge is probably the most significant product developed for the C64. It contains a full 64K COMAL system plus complete bank switching circuitry. This is equivalent to 8 game cartridges, utilizing the latest technology. And while just a plain 64K RAM expansion cartidge is being advertised for $370, the 64K COMAL Cartridge is less than half that - only $128.90 complete with 2 books and 2 demo disks.

The selling price of the cartridge is expected to remain at $128.90 or possibly even increase. Commodore is manufacturing the cartridge in Denmark rather than Hong Kong. This has many advantages including better quality control, safety, and much faster delivery. The disadvantage is that production costs are about TRIPLE. We applaud the decision. A top quality product deserves top quality treatment.

The cartridge currently is being distributed in EPROM form. The ROM version will not be available until Spring 1985. Both EPROM and ROM versions are identical except for the copyright notice on power up. Both also have the same selling price. But, only the EPROM version is available now, and IS shipping! But be warned our stock is low and when we run out it may be a month delay before we get more in.

OFFER EXPIRED

Our offer of 2 free disks and 2 free books with each cartridge has expired. The C64 COMAL 2.0 Cartridge is now available in limited quantities and is shipping on a first come first served basis. All orders processed by December 10, 1984 have been filled. Over 500 people received the FREE books and disks before the offer expired. It was our thank you for their early suppot.

NEW PRICES

The cartridge is still available separately for $99.95 (the same price as before). Since it is just a plain cartridge, with no manual or examples, it is meant only for schools and user groups. The cartridge support package includes the COMAL HANDBOOK Second Edition, Cartridge Graphics and Sound book, and two Cartridge DEMO Disks (#1 & #2). It is available separately for $65 (a $70 value). The current price for the DELUXE CARTRIDGE PAK is $128.90. It includes both the cartridge and the cartridge support package. If you want to get the cartridge - order the DELUXE CARTRIDGE PAK at $128.90.

HIGHER PRICES IN 1985

Currently we are not making much money on the sale of the cartridge (we actually LOST money on the first 500 sold). Our goal is to make it available to as many people as possible, thus we keep the price as low as possible. The price may have to be increased during 1985, so get yours before then. And no discounts are available due to the small markup.

Ask anyone who has a COMAL cartridge. They'll confirm that it opens a whole new world for you.

EMPTY SOCKET-LESS CARTRIDGES

The first 1000 COMAL 2.0 cartridges shipped in North America were EPROM cartridges and did not have an empty socket. When Commodore finally produces the ROM cartridges, they are supposed to have an empty socket in them. It is very unfortunate that all the delays with Commodore occurred. However, the COMAL is identical in both the EPROM and ROM cartridges (except for the copyright notice on the start up screen). Thus it comes down to this: is the empty socket useful to YOU.

The empty socket allows you to insert an extra EPROM, made specifically for that socket. It contains a specially prepared COMAL 2.0 program, such as a REAL ESTATE SERVICE program. With the extra EPROM in place, EVERY time the computer is turned on it will run that program. It will not come up in COMAL! This is very useful for specific applications - and the programs for them can be written right now to prepare for the day the cartridges with the empty socket become available.

Luckily, the vast majority of the people getting the cartridges now will not need the empty socket. However, a few just may. One person has asked us if we will allow them to 'trade in' the EPROM cartridge for a new ROM cartridge. It doesn't look like we will be able to do that directly. However, we are sure, that there are plenty of people who will be buying the new ROM cartridges later who would not mind 'trading' cartridges for an EPROM version. As a matter of fact, there may be quite a few who would prefer to have an EPROM version (so they can change part of the COMAL system for their specific purposes). We may be able to help with such 'trades' but, hope it can all happen locally with a person to person trade.

SPANISH COMAL

Yes, out of New Mexico comes SPANISH COMAL. It is the same COMAL 0.14 that we all know, but the keywords are in SPANISH. Of course, so are the error messages. This version of COMAL goes against what we've always been saying. This version is only compatible with COMAL 0.14 **SAVE / LOAD** files. The LIST files are not compatible. This is a specialized version not available on any of our disks. If you know anyone who knows Spanish better than English, and has a Commodore 64 with 1541 (or compatible) disk drive, you are invited to try out SPANISH COMAL. Send $10 for the special disk (not available via the 800 number).

CARTRIDGE DEMO DISK 3 STORY
or ALL AT ONCE2 error

When we first did the master disks for CARTRIDGE DEMO DISKS #1, #2, #3, and #4 we had all the files in alphabetical order on the disk. After a few weeks, we thought that perhaps it would be nicer for users if they were grouped by category first, and then put in alphabetical order. So we redid the disk masters. However, adding the category identifiers (ie, ---SHAPE FILES--) used up a few blocks of space on the disk. And disk #3 had no extra space. But we remembered the advice about cleaning up the programs name table: If you SAVE a program to disk, it is saved along with its current name table. If you LIST it to disk, only the program is stored on disk. Then when you ENTER it back, the name table is recreated. If the program had been revised a few times, there probably were names in the name table which no longer were used. They took up space needlessly. ENTERing the program cleared them out, resulting in more free memory, and possibly 1 less block when SAVEd to disk. We did this for many of the programs on CARTRIDGE DEMO DISK #3 and got enough blocks free on the disk to add the category identifiers. However, we were bit by one of the new features of COMAL 2.0: LINKED SPRITE SHAPES!

Yes, it is possible to define sprite shapes and then link the images onto the program using the LINKSHAPE command. Then the part of the program that defined the images can be deleted. A very nice touch! When you SAVE the program the images are stored with the program, and come back again with a LOAD command. But, if you LIST the program to disk the shape images are no longer attached to the program. But if you ENTER it back again you may not notice it since the images were still in the computer from before. Thus we lost the shape images from the program ALL'AT'ONCE2. The unfortunate part was that this was our second master for DISK #3. Several people received disks made from this flawed master. We noticed the error after about a week and immediately corrected the master and redid all the disks in stock. Thus our DISK#3 was updated a third time. If you were one of the unlucky ones to get a flawed ALL'AT'ONCE2 program we will give you a new DISK #3 if you return the flawed one (no charge of course).

But you CAN fix this yourself, and this gives me a perfect opportunity to give step by step directions on how these fun ALL'AT'ONCE programs are developed - and how you can fix the missing shape images problem.

ALL AT ONCE - GRAPHICS, SPRITES, SOUND

There is a different ALL'AT'ONCE program on Cartridge DEMO DISKS #2, #3, and #4. Each program demonstrates COMAL controlling sprites (at fast speeds), playing music (in multiple voices) and drawing patterns (turtle graphics). The amazing thing is that it is doing all this at one time - fast! People find it hard to believe that no tricks are needed to do this, it is all built into the COMAL 2.0 Cartridge. Now, here is how these ALL'AT'ONCE programs are developed:

Write three separate programs. One that draws a nice pattern on the screen, one that moves sprites around using the built in ANIMATE features, and one that plays music using the built in automatic music system. The notes for the music can be conveniently stored as a sequential data file, and then read into the array when the program is run (which is how we did our programs). Divide each program into two parts, initialization and processing. LIST the sound and sprites programs to disk (so that we can use MERGE later).

Now, merge the sprites program onto the end of the turtle graphics pattern program (use the MERGE command). Since the sprites and sound are "interrupt driven" they run in the "background" - simultaneously with the main program (the turtle pattern program in this case). The graphics proram is the controlling part of the ALL'AT'ONCE program. Find where the graphics program repeats the most (hint: look in FOR ... ENFOR and other loops) and add a line to check the sprite processing procedure (ie, CHECK'SPRITES). Make sure that the sprite procedure doesn't take too long (just check if any sprites have stopped moving - and if so start them up again). Also, add an IF statement to restart the music if has finished playing.

You can then LINK the sprite shapes to the program and they will be saved with the program. (You must RE-LOAD and RE-LINK the shapes if you ever LIST the program to disk and ENTER it back again).

That's it. Impressive results. No tricks.

Now, I promised to tell you how to fix the ALL'AT'ONCE2 program missing its sprite shapes:

1) Put your **COPY** of Cartridge Demo Disk #3 in the disk drive (NEVER write on your master copy of ANY disk).
2) LOAD the target program:

```
LOAD "ALL'AT'ONCE2"
(don't run the program yet)
```

3) Now we will use LOADSHAPE commands to load the sprite shapes from the disk files on the disk. These files are at the end of the disk. To see them issue the command:

```
CAT
```

4) Now we will be using some sprite commands, so we must tell COMAL to use the sprite package:

```
use sprites
```

5) Next we will load the proper sprite images into memory with the following three commands (press the RETURN key after each command):

```
loadshape(0,"shap.bats0")
loadshape(1,"shap.bats1")
loadshape(2,"shap.bats2")
```

6) Now the sprite images are in memory, but they have not been linked to the program. This is done with the LINKSHAPE command. Type in the following commands:

```
linkshape(0)
linkshape(1)
linkshape(2)
```

7) The program has now been fixed, so we will now update the disk (remember, do this to a COPY, NEVER write to a master disk). The first two lines of all the programs on the cartridge demo disks contain the DELETE and SAVE commands needed. Type the following command:

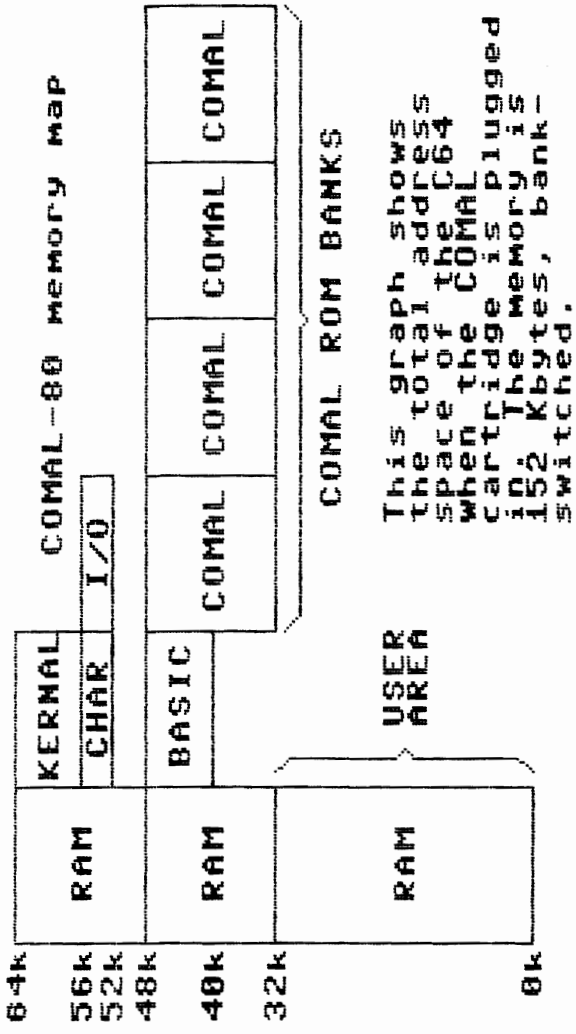
```
list -20 <press RETURN>
```

8) You now see the following lines:

```
0010 // delete "0:all'at'once2"
0020 // save   "0:all'at'once2"
```

9) Move the cursor up to line 10 and press the space bar until the line numbers and the two slashes are erased. Make sure the write protect tab is off the disk and press the RETURN key. The old copy of the program will be erased.

10) When the disk drive is finished erasing the old copy, the cursor will appear on the next line (line 20). Again, press the space bar until the line number and the two slashes are erased. Press the RETURN key now and the new copy of the program will be saved to disk.



COMAL IMPLEMENTATIONS

as of September 1984

Compiled by:

Kevin Ryan
Secretary of the Comal Standards Group,
Department of Computer Science,
Trinity College
Dublin 2
IRELAND.

1) ACORNSOFT
   Hardware: BBC Micro A & B
             Acorn Electron
   Operating System: BBC MOS
                     Electron MOS
   Contact:
     Acornsoft
     Betjeman House
     104 Hills Road
     Cambridge CB2 1LQ
     ENGLAND
     +44-223-316039

2) COMMODORE
     SEE Unicomal

3) DANSK DATA ELEKTRONIK
   Hardware: SPC/1
             Supermax
   Operating System: Mikados
                     Supermax OS
                     Unix
   Contact:
     Danish Data Elektronics a/s
     Herlev Hovegade 199
     DK-2730 Herlev
     DENMARK
     +45-2-845011

4) INSTRUTEK
     See Unicomal

5) METANIC APS
   Hardware: All Z-80 Systems with CP/M
             and at least 48K including:
             Apple with CP/M card
             Osborne
             Kaypro
             S100 Z-80 Systems
   Operating System: CP/M
   Contact:
     Mogens Pelle
     Metanic Aps
     Byvej 11
     DK-3600 Stenlose
     DENMARK.
     +45-2-172728

6) REGNECENTRALEN
   Hardware: RC Piccolo
             RC Partner
             RC Piccoline
             RC 855
   Operating System: CP/M 80
                     Concurrent CP/M-86
   Contact:
     Torsten Schmidt
     A/S Regnecentralen of 1979
     Lautrupbjerg 1
     DK-2750 Ballerup
     DENMARK
     +45-2-658000

7) TELI DATORER
   Hardware: Compis
             Scandis
   Operating System: CP/M-86
                     MS-DOS
                     Unix
                     Zenix
   Contact:
     Teli Datorer
     Box 213
     S-14901 Nynashamn
     SWEDEN.
     +46-7-5262390

8) TCD COMAL
   Hardware: VAX
             PDP 11
   Operating System: VAX/VMS
                     RSTS/RSX
   Contact:
     Rody Ryan
     Software Engineering Laboratory
     Dept. of Computer Science
     Trinity College
     Dublin 2
     IRELAND
     +353-1-772941

9) UNICOMAL
   Hardware: CBM 4032
             CBM 8032
             CBM 8096
             SUPERPET
             Commodore 64
             Plug in board for PET
              and CBM
   Operating System: Commodore OS
   Contact:
     Jens Erik Jensen
     Unicomal Aps.
     Snogaardavej 10C
     DK-2820 Gentofte
     DENMARK.
     +45-1-651762

Contact:
Jan Nyman
Commodore Data
Bjerrevej 67
DK-8900 Horsens
DENMARK
+45-5-64155

Contact:
Instrutek
Christiansholmsgade
DK 8700
Horsens, Denmark

Editors Note: We are adding the following versions which have been announced for early 1985 release after Kevin compiled his list:

10) IBM
Hardware: IBM PC
IBM PC XT
IBM PC AT
Operating System: MS DOS
PC DOS
Contact:
IBM Denmark

11) MYTECH
Hardware: 128K Personal Computer
WICAT
IBM PC
IBM PC XT
IBM PC AT
TELI Compis
Operating System: MS DOS 2.0 and up
CCP/M
CP/M
CP/M 86
Unix version 7
and up
Contact:
Bo Gardmark
Mytech Data AB
Box 7230
S-402 35
Goteborg, Sweden

A MISSED IBM PC SALE

We were pleased to hear from a computer salesperson who informed us that a customer was almost ready to buy a complete IBM PC system, but when told that it couldn't run COMAL decided not to get it. Fortunately, that problem will soon be solved, since both Mytech and IBM have announced COMAL systems to run on the IBM PC.

IGNORED AGAIN

COMAL has been around for over 10 years now. Several years ago COMPUTE! magazine had a nice article about COMAL. Then their November 1984 issue had another article on COMAL (referred to as SUPERBASIC). So, obviously COMPUTE! knows about COMAL. It was surprising that in their January 1985 issue COMAL was not even mentioned in the article titled: Which Computer Language Is Best?.

The article gives three criteria to judge a language:

1) Suitability of the language to the task
2) Ease of learning and using the language
3) Availability of the language on the computer we want to use

The article was talking about a programming language for home and personal use. This is precisely what COMAL was designed for. It is suitable for that use, easy to learn and use, and readily available.

It was even more interesting that in the same issue, COMPUTE talks about MSX BASIC. They show a benchmark program used with 14 different computers. Then they point out how MSX BASIC is faster than all except the IBM PC. Why was COMAL left out of the comparison? We ran the benchmark test on the Commodore 64 with the COMAL 2.0 Cartridge. It was nearly twice as fast as the MSX BASIC, and four times faster than the TI-99/4A BASIC.

The idea behind MSX BASIC was to have a compatible language between different machines. That is also the idea behind COMAL. Now, that IBM announced that they will have a COMAL 2.0, COMAL is available on the following machines:

<=== SEE PREVIOUS LISTING <===

(includes Apple, Commodore, CP/M and IBM)

COMAL is standardized. COMAL is here to stay. Already 5 European countries have adopted COMAL as the language to be used in their schools. Welcome COMAL.

JUST FOR BEGINNERS

by Captain COMAL and Friends
(special friend for this: Phyrne Bacon)

Boy, there are lots of technical articles in this issue. But fear not! Beginners are not forgotten. I received a very nice booklet called **HINTS ON PROGRAMMING IN COMAL** by Phyrne Bacon. She wrote it for the Gainesville Commodore Users Group COMAL SIG. It was a great idea and inspired the beginners column for this issue. If you would like to contribute ideas and material JUST FOR BEGINNERS please send it in.

Here are some HINTS:

1) COMAL DISKS
There are many COMAL disks. Most of them have the COMAL system right on the disk. The COMAL is the same on each disk - only the programs are different. Put write protect tabs on all your MASTER COMAL disks. The disks can and should be copied. Then use only the copies. Put write protect tabs on any of the copies that you only wish to LOAD from (and not write to).

2) START UP
Before you can do anything in COMAL you must first convert your computer from BASIC into COMAL. Do this by LOADing and RUNning the COMAL system. To do this take a COMAL disk, insert it into the disk drive and enter this command:

```
LOAD "BOOT*",8
```

A short start up program should load into the computer. Now enter this command:

```
RUN
```

The screen should fill with some introductory COMAL information. You will have time to read it since it takes about 90 seconds to LOAD in the COMAL system.

3) ERROR MESSAGES
COMAL can provide nice error messages, but the messages themselves are not built in. They are in a file called COMALERRORS. You should have a disk in the drive with this file on it. You can put this file on any disk by running the COMAL program called GENERATE ERROR (on the COMAL SAMPLER DISK). Or better yet, you can use a special program on TODAY DISK #4 called RAM'ERRORS. It reads the error file and keeps an exact copy in the computer. Then you no longer need the disk based file. This is great since error messages are then FAST (no disk access delay). And schools will like it since several computer can be hooked into one disk drive and not need that special file. However, there is one small drawback - when the error messages are in RAM you are not allowed to use sprite images 15 thru 35 (you may still use sprite images 0-14 more than enough for most programs). To make it easier for you, this RAM ERROR system is built into the HI program on the TODAY DISK #4 and #5.

4) HI PROGRAM
When COMAL starts up it looks for a COMAL program on the disk called HI. It then LOADs and RUNs that program. Most COMAL disks have a HI program. Most HI programs are alike in some ways, but specialized in others. The HI program on TODAY DISK #5 includes quite a few things all automatically for you.

5) DISK COMMANDS
To see what is on the disk in the drive, issue the command:

```
CAT
```

To slow down the catalog (directory) hold down the CONTROL key. To stop it hit the STOP key.

To send a DOS command (as listed in the disk drive users manual) to the disk drive use the PASS command:

```
PASS "R:NEW'NAME=OLD'NAME"
  (renames a file)

PASS "I"
  (initializes the disk in the drive)
```

To delete (scratch) a file from the disk use the DELETE command:

```
DELETE "0:NAME"
```

To see the status of the disk drive:

```
STATUS
```

6) LOADING AND RUNNING PROGRAMS
A good program to start with is the HI program on most COMAL disks:

```
LOAD "HI"
```

then type:

```
RUN
```

Another way to do both LOAD and RUN commands at once is with the CHAIN command:

```
CHAIN "HI"
```

LOAD and CHAIN only work with PRG type files (not SEQ type) that were created by the COMAL 0.14 SAVE command.

7) STOP A PROGRAM
To stop any program press the STOP key. If that does not work press both the STOP and RESTORE keys at the same time.

8) RUNNING SEQ TYPE PROGRAM FILES
SEQ type COMAL files can be several things including DATA files and COMAL program files. If it is a DATA file you can't run it (if its name ends with .DAT or .TXT it is probably a DATA file). If it is a COMAL program you can run it like this:

```
Enter command:  NEW
Enter command:  ENTER "NAME.L"
Enter command:  RUN
```

If nothing happens when you say run, it might be only a partial COMAL program, a segment or module called a procedure or function. These files are meant to be added onto another COMAL program. After using COMAL for awhile you may wish to learn about them.

9) LISTING PROGRAMS
To see a listing of a program, type:

```
LIST
or
EDIT
```

LIST gives a listing of a program with indentations. The indentations are especially handy for studying the programs structure. EDIT gives a listing without the indentations (more like the BASIC LIST), and is easier to use when editing programs since you then don't have to delete the indentation spaces. To list only part of a program use the same method as with BASIC:

```
LIST 20-70
EDIT -200
LIST 400-
```

Don't forget the space after the word LIST or EDIT. COMAL is particular about spaces.

10) PRINT A PROGRAM LISTING ON PRINTER
To print a program listing on your printer issue these commands (the LP stands for Line Printer):

```
SELECT "LP:"
LIST
```

To select lower case mode on your printer (using secondary address 7 in this example):

```
OPEN FILE 255,"",UNIT 4,7
SELECT "LP:"
LIST
```

11) DELETING LINES
To delete line 80 type:

```
DEL 80
```

Don't forget the space. To delete a block of lines, for example:

```
DEL 30-90
DEL -40
```

12) SAVING PROGRAMS
First you need a disk to store the programs on. To prepare your disk, get a BLANK disk (a new empty disk) and format it. To format a disk, insert it into the disk drive and issue the command:

```
PASS "N0:WORKDISK,WD"
```

The WD at the end is the ID of the disk. Make the ID of each of your disks different (ie, use AW and RV and IX etc).

There are two ways to save a COMAL program:

```
SAVE "MINE"
 or
LIST "MINE.L"
```

SAVE produces a program file (PRG). LIST produces an ASCII sequential (SEQ) file. Most of the time you should use SAVE (and LOAD to get it back later).

Make sure that the write protect tab is NOT on the disk when you try to save your program on it. Also make sure it has enough blocks free to fit your program.

13) COMAL COMMANDS
A complete HELP file that lists all the COMAL COMMANDS is on some of the COMAL DISKS. An option called HELP is even included in some of the HI programs (such as on the SAMPLER DISK).

COMAL USERS GROUP includes a handy printed listing of these commands with each disk you order from them. Or to get a free copy send a self addressed envelope (business size) and ask for the POCKET QUICK GUIDE. Feel free to copy it for friends.

Short explanations of the commands are in the book: COMAL FROM A TO Z. It includes about a paragraph of explanation for each COMAL COMMAND.

For a complete detailed presentation of COMAL COMMANDS get the book: COMAL HANDBOOK.

But to start with, beginners should use a tutorial book. BEGINNING COMAL is the favorite book (so popular it sold out - it is being reprinted now in ENGLAND and should be available in early February 1985). And a new book, GRAPHIC PRIMER, is a beginners guide to graphics and sprites.

14) PROGRAMMING
Use only one COMAL COMMAND per line. Most BASIC commands work in COMAL.

15) PROCEDURES
A procedure is like a subroutine. As an example, we might have a procedure called PAUSE as follows:

```
600 PROC PAUSE
610 FOR DELAY=1 TO 999
620 NEXT
630 ENDPROC
```

Now, whenever we want to use this routine, I would have a line like:

```
140 PAUSE
  or
210 IF Z=0 THEN PAUSE
```

You can't include spaces in a procedure name (ASK NAME is not accepted), however they can include apostrophes (ASK'NAME is fine). Procedures is one of the nicest things about COMAL.

16) AUTOMATIC LINE NUMBERING
When you type in your program COMAL will put in the line numbers for you. To begin this auto line numbering type:

```
AUTO
```

or to begin with line 500, type:

```
AUTO 500
```

To quit AUTO mode hit the RETURN KEY twice.

COMAL 2.0 INPUT FIELD

One of the added professional features of COMAL 2.0 is the protected INPUT field. The simple INPUT statement now automatically has a protected field. And you can specify the exact length of it too using the INPUT AT statement:

```
INPUT AT r,c,max: prompt$: reply$

r   => row of first character
c   => column of first character
max => max num characters for reply

INPUT AT 5,1,6: "NAME=>": NAME$
```

This prints the prompt, NAME=> so that the N is at row 5 column 1. It also sets up the input field to be 6 characters long. If you specify the field length to be 0, COMAL will only accept the RETURN key as a valid response:

```
INPUT AT 5,1,0: "HIT RETURN": REPLY$
```

OVER 1,000 PROGRAMS

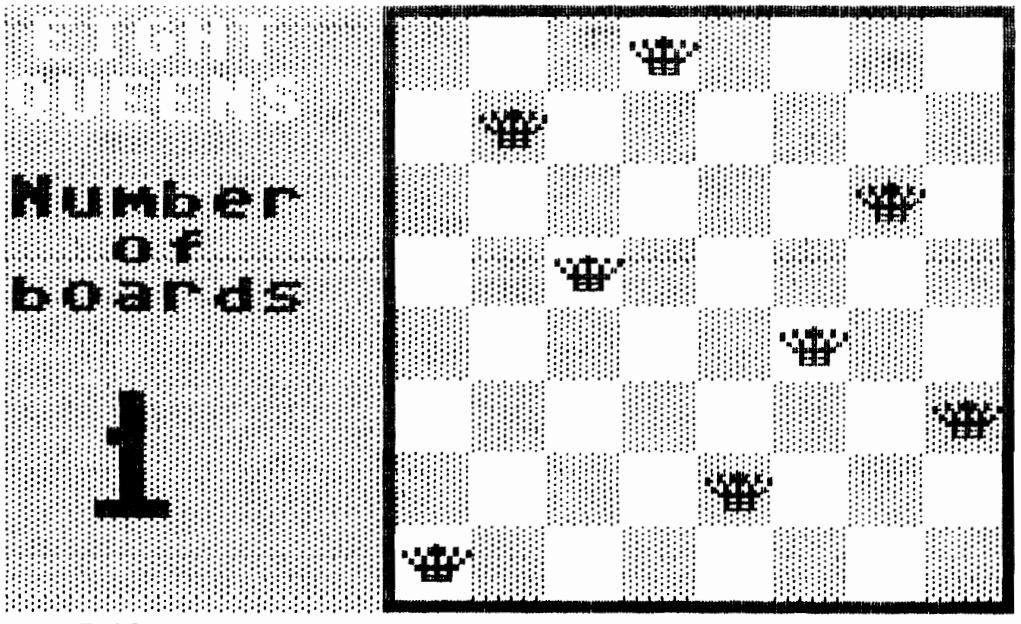
We have over 1,000 C64 COMAL programs on 30 different disks. We created a database of the programs on the first 20 COMAL 0.14 disks including the program name, author name, disk name, type (SEQ or PRG), comments, category, and blocks. We numbered the programs as well. Our report is 22 pages long, 1063 files listed. Send $2 for your copy (ask for the 22 page, 1063 files listing). A reduced copy of the first report page is included here. With so many disks and programs, we frequently are asked what programs are on each disk. We hope to list the directory for all 30 of our disks in this issue.

SPECIAL OFFER: Right now we are making a special offer to any school or user group (or anyone wishing to start a user group). A complete set of the first 19 COMAL 0.14 disks for only $94.05 (less than $5 each). Ask for the **19 DISK SET.** No substitutions, complete set only.

NUM	PROGRAM NAME	AUTHOR NAME	DISK NAME	TYPE	COMMENTS	CATEGORY	BLOCKS
1:	1525 pixel dump	loren wright	user group disk #2	SEQ	textscreen pixel dump	utility	16
2:	1525 screen dump	eric giguere	user group disk #1	PRG	7 bit graphics screen dump	utility	11
3:	3d'cube		user group disk #1	PRG			17
4:	7-3hill.hrg	unknown	slide show #2	PRG	hi res picture	data file	32
5:	8023p'options4	david maven	utility disk #1	PRG	set print options for cbm 8023	utility	9
			today disk #2	PRG	set printer options on cbm8023	printer	9
6:	8023p'options4.l	david maven	today disk #2	SEQ	set print options for cbm 8023	printer	12
7:	84expenditures	peter foiles	user group disk #3	PRG	small data base for home manag	data base	30
8:	a maz'in basic.b	mike miller	user group disk #3	PRG	joystick maze program (basic)	game	6
9:	a maz'in comal.c	mike miller	user group disk #3	PRG	joystick maze game(comal 0.14)	game	14
10:	a maz'in simon.s	mike miller	user group disk #3	PRG	joystick maze game (simons)	game	5
11:	abc.sprite	peter foiles	utility disk #1	SEQ	sprite data file - all abc's	data file	7
			today disk #4	SEQ	sprite data file - all abc's	data file	7
			user group disk #3	SEQ	sprite data file - all abc's	data file	7
			auto run demo disk	SEQ	abc's as sprite images	data file	7
12:	abs	len lindsay	comal handbook disk	PRG	program from the book	book prog	1
13:	accept'demo	robert shingledecker	user group disk #4	PRG	input fields	demo	10
14:	addition	borge christensen	beginning comal disk	SEQ	program from the book	book prog	4
15:	address	joan ware	user group disk #4	SEQ	data file for record keeper	data file	1
16:	addresses	borge christensen	beginning comal disk	SEQ	program from the book	book prog	1
17:	addrpgr	borge christensen	beginning comal disk	SEQ	program from the book	book prog	6
18:	albert.hrg	unknown	slide show #2	PRG	albert einstein hi res picture	data file	32
19:	alpha1'gen	kevin quiggle	today disk #4	PRG	gutenburg data file creator	utility	24
			user group disk #2	PRG	gutenberg data file creator	utility	24
20:	alpha1.dat	kevin quiggle	today disk #4	SEQ	gutenburg character data file	data file	21
			user group disk #2	SEQ	gutenberg character data file	data file	21
21:	alpha2'gen	kevin quiggle	today disk #4	PRG	gutenburg data file creator	utility	30
			user group disk #2	PRG	gutenberg data file creator	utility	30
22:	alpha2.dat	kevin quiggle	today disk #4	SEQ	gutenburg character data file	data file	30
			user group disk #2	SEQ	gutenberg character data file	data file	30
			auto run demo disk	SEQ	gutenberg character data file	data file	30
23:	and	len lindsay	comal handbook disk	PRG	program from the book	book prog	3
24:	angry.dragon	valerie kramer	user group disk #3	PRG	animated picture of dragon	graphics	18
25:	another'moire	unicomal	user group disk #3	PRG	moire drawing	graphics	8
26:	append	len lindsay	comal handbook disk	PRG	program from the book	book prog	1
27:	appendix'd	len lindsay	comal graphics	PRG	program introducing proc/func	demo	2
28:	aprilfool	unicomal	user group disk #1	PRG	good multi color picture	graphics	9
29:	Arabesque	colin thompson	auto run demo disk	PRG	symetrical patterns	graphics	5
30:	arabesque1	colin thompson	user group disk #4	PRG	patterns	graphics	9
31:	arabesque2	colin thompson	user group disk #4	PRG	patterns	graphics	10
32:	arabesque3	colin thompson	user group disk #4	PRG	patterns	graphics	8
33:	arabesque4	colin thompson	user group disk #4	PRG	patterns	graphics	9
34:	arabesque5	colin thompson	user group disk #4	PRG	patterns	graphics	8
35:	arabesque6	colin thompson	user group disk #4	PRG	patterns	graphics	9
36:	arabesque7	colin thompson	user group disk #4	PRG	patterns	graphics	10
37:	arabesque8	judy nahman	user group disk #5	PRG	draws patterns	graphics	12
38:	arcs.demo	mindy skelton	user group disk #5	PRG	draws an arc	demo	4
39:	asc.func	unknown	user group disk #5	SEQ	returns the ascii value	function	1
40:	atari graphics	m zuanich/s lipscomb	user group disk #3	PRG	converted from atari graphics	graphics	11
41:	atn	len lindsay	comal handbook disk	PRG	program from the book	book prog	1
42:	auntie	borge christensen	beginning comal disk	SEQ	program from the book	book prog	17
43:	auto'directory	captain comal	utility disk #1	PRG	present a disks directory menu	utility	13
44:	back	len lindsay	comal graphics	PRG	turtle demo	demo	3
45:	background	len lindsay	comal graphics	PRG	graphics demo	demo	3
46:	background.l	donald pipkin	today disk #2	SEQ	returns current background col	function	1
47:	bagel	colin thompson	user group disk #3	PRG	string art type pictures	graphics	11
48:	basic'to'comal	unknown	utility disk #1	PRG	first step in basic to comal	utility	6
			user group disk #2	PRG	convert a listed basic program	utility	6
49:	beep/gong.demo	joan ware	user group disk #3	PRG	sound demo	music	8
50:	beeper	joan ware	user group disk #1	PRG	sound demo	music	2
51:	beeper.l	joan ware	utility disk #1	SEQ	makes a beep noise	utility	2
52:	benchmark64.basl	lindsay/kortendick	today disk #2	PRG	benchmark timing prog (basic)	utility	26
53:	benchmark64.coma	lindsay/kortendick	today disk #2	PRG	benchmark timing prog (comal)	utility	27
54:	benchmark64.l	lindsay/kortendick	today disk #2	SEQ	benchmark timing prog (comal)	utility	37
55:	big'letter	greg beaumont	today disk #1	PRG	draws big letters with turtle	utility	4
56:	big'letter.l	greg beaumont	today disk #1	SEQ	listed to disk version	utility	5

EIGHT QUEENS
A CLASSICAL PROBLEM

If you are a chess buff, you have heard of the EIGHT QUEENS problem. A QUEEN is a playing piece in chess. It can take other pieces in its path diagonally, vertically, or horizontally. The problem is to place 8 queens on the chess board so that none of them can 'take' any of the others.
(continued next page)



(EIGHT QUEENS - continued from prev page) People familiar with this problem, also may know that many computer solutions have been published, where the computer program will print all possible solutions. One of those programs was written in PET COMAL years ago. But, this program is one better than all the others. It actually shows you how the computer is thinking about placing the queens on the board. We ran the program a few times at several COMPUTER SHOWS and alot of people really enjoyed it. Now, you can watch it yourself. A version for COMAL 0.14 is on the TODAY DISK #5. The version for COMAL 2.0 is on Cartridge Demo Disk #2 and even includes an option to print the solutions on the printer using the built in graphics screen dump. (See the example printout).

THE MISSING COMMA

Charles Rice was one of the first COMAL 2.0 users. He wrote us about a possible bug in COMAL 2.0. Here is what he discovered:

```
PRINT ""19""     cursor at row 2 col 1
PRINT ""147""    cursor at row 2 col 1
PAGE             cursor at row 1 col 1
```

He thought all three should move the cursor to the same location - and he is right! First, I should explain the nice way COMAL 2.0 allows us to print a CHR$ or control code. PRINT ""19"" is the same thing as PRINT CHR$(19). Thus the first example will do a HOME CURSOR, the second a CLEAR SCREEN, and the third a CLEAR SCREEN. Now, look at the examples again. Do you see what is missing in the first two? It is common practice to clear the screen with PRINT CHR$(147). This is quite acceptable for most purposes. However, as Charles points out, you do end up on line 2 not line 1. So if you want all three examples above to end up in the same place, add a comma at the end of the first two. A normal PRINT statement moves to the next line when done (i.e., line 2). Put a comma at the end to supress the line feed. So the corrected examples are:

```
PRINT ""19"",    cursor at row 1 col 1
PRINT ""147"",   cursor at row 1 col 1
PAGE             cursor at row 1 col 1
```

HOW COMAL STATEMENTS ARE STORED

by John H McCoy
1615 Bagwell
Huntsville, TX 77340

The COMAL translator translates each statement into a tokenized internal form as it is entered at the keyboard, or, from a file via the ENTER command. A token is a short, preferably a single byte, code used to represent an operaton to be performed. The operation of assigning a value to a variable, for instance, will be tokenized into one of three tokens. $3A is used when the variable is a real, $3B is used when it is an integer and $3C when it is a string variable.

The form of the internal language used by COMAL is called reverse polish notation. It is probably so called because us foreigners can't pronounce Lukasiewicz, the name of the Pole who developed the technique in the 1920's. To give due credit, I shal call it reverse Lukasiewicz notation but refer to it as RLN for short. Sorry, but I can't pronounce it either.

In mathematical expressions, arithmetic operations are called operators while the things they operate on are called operands. Thus in the expression AMOUNT/5, AMOUNT and 5 are operands while "/" is the division operator. Similarly, when dealing with the string expression, STRING1$+"ABC", STRING1$ and "ABC" are operands and "+" is the symbol for the operation of concatenation (joining together) the two string values. In standard use most operators are placed between their operands. The principle of RLN, however, is that the operator is written after its operands. Thus AMOUNT/5 in standard notation becomes AMOUNT 5 / in RLN. Readers who use HP calculators (except the latest model which has switched to standard arithmetic notation) will be acquainted with this form of notation.

The reason for adopting RLN is to eliminate the ambiguity that arises when a statement such as 5*AMOUNT-2 is written. It is possible to code the translator to deduce "correctly" the order the operations are to be done, but

it significantly reduces the work of interpretting the statements at run time if the statements are translated into a non-ambiguous form. RLN meets this requirement. In addition RLN is sufficiently close to the form of the statements as they were entered that the original statements do not have to be kept. They can be reconstructed at will with the LIST or EDIT command.

RLN COMAL Statement Format

A translated COMAL statement format is:

```
STMNT'NO  LNTH  BODY
```

STMNT'NO is the statement number coded as a 2 byte integer;
LNTH is a 1 byte integer statement length shich includes the STMNT'NO and LNTH bytes;
BODY is the body of the COMAL statement in RLN.

```
COMAL STATEMENT
---------------
0010 N:=49152
0020 NUM#:=32767

STMNT'NO LNTH BODY
-------- ---- -------------------------
00 0A    0C   07 01 01 90 40 00 00 00 3A
00 14    09   08 02 02 7F FF 3B
```

COMAL TABLES

Deciphering the statement number and length are straight forward if one is handy with hexadecimal arithmetic. Deciphering the body of the statement, however requires some familiarity with how COMAL does things.

COMAL uses numerous tables. Some of these tables it uses in recognizing COMAL words and in translating, reconstructing and interpreting the COMAL statements. These tables are a part of the COMAL code. Another set of tables are those it constructs to set up and keep track of space for the variables and procedures used by the program.

At this time I think it safe to say that only the original authors know where all the tables are located, their use and descriptive titles. Through reverse engineering it is perhaps possible to locate the tables and determine their use. One can, however, only speculate as to the original title given to a table.

Tables in the COMAL code appear to fall into three primary types. Those used for code translation; those used as indexes (pointers) into other tables; and those used for jump addresses. There are exceptions. There are also some groups of addresses which seem to be used as switches and might collectively be considered as tables.

In most instances where the table contains jump addresses, the jump is effected by using RTS. In such cases, a +1 will have to be added to the table address to get the actual jump address.

COMAL TABLES
(Best Guess as of 11/03/84)

```
TABLES LOCATED IN COMAL

$0814-$0BCB COMAL KEYWORD TABLE
            1 BYTE LENGTH FOLLOWED BY
             KEYWORD
            177 ENTRIES OF WHICH 6 ARE
             NULL (NULL ENTRIES COUNT IN
             DETERMINING POSITION)
$0BCC-$0C7D ??
$0C7E-$0D2F INDEXES
$0D30-$0DE1 CODES
$0DE2-$0EA2 CODES
$0EA4-$0F66 INDEXES
$0F67- ??   INDEXES
  ?? -$104B SWITCHES
$104C-$10CB CHAR TYPE CODES

$13C0-$13D1 HI BYTE
$13D1-$13E2 LO BYTE
            (RTS ADDRESS FOR $11A7)
            (INDEXED BY $0EA4-$0F66)

$1796-$179E HI BYTE
$179F-$17A7 LO BYTE
            (RTS ADDRESS FOR $172E)
            (INDEXED BY $0C7E-$0D2F)

$17D3-$17F0 HI BYTE
$17F1-$18E0 LO BYTE
            (RTS ADDRESS FOR $1428)
            (INDEXED BY $0F67-??)

$1888-$188C HI BYTE
$188D-$1891 LO BYTE
            (RTS ADDRESS FOR $183B)
```

```
$1A50-$1A56 MSG: "ERROR"

$2C9C-$2CCA NOT A TABLE
            KEYWORD LOOKUP ROUTINE

$38CA-$38CE "AT" USED BY $7CC1

$4776-$4785 CONSTANTS USED BY $476B

$798F-$7A89 HI BYTE
$7A8A-$7B84 LO BYTE
            (RTS ADDRESS FOR $4392)
            (INDEXED BY KEYWORD POS)

$7B85-$7BD8 HI BYTE
$7BD9-$7C2C LO BYTE
            (RTS ADDRESS FOR $663C)
            (INDEXED BY KEYWORD CODE)
            (PRESENTLY KNOWN CODES)
            (EDIT -- 4; LIST -- $15)

$7C81-$7C89 "END AT" USED BY $7C7B
$7C8A-$7C93 "STOP AT" USED BY $7C97

$7F57-$8051 INDEXES
$8052-$814C CODES
$814D-$818B INDEXES
$818C-$81CA CODES

$84E4-$8522 HI BYTE
$8523-$8561 LO BYTE
            (RTS ADDRESS FOR $822C)
            (INDEXED BY $7F57-$8051)
            (        OR $814D-$818B)
```

TABLES CREATED BY COMAL

USER DEFINED NAMES
(VARIABLES, PROCS AND FUNCS)

```
START PTR $003C HI BYTE
          $003D LO BYTE
END PTR   $003E HI BYTE
          $003F LO BYTE
FORMAT:  1 BYTE LENGTH FOLLOWED BY NAME
ENTRIES: 1 ENTRY FOR EACH DIFFERENT NAME
```

NAME ALLOCATION TABLE

```
START PTR $003A HI BYTE
          $003B LO BYTE
END PTR   $003C HI BYTE
          $003D LO BYTE
FORMAT: 1 BYTE TYPE CODE
        $00 NOT AVAILABLE
        $10 REAL
        $11 INTEGER
        $12 STRING
        $13 LABEL
        $14 PROCEDURE
        $15 FUNCTION
        (ADDITIONAL BITS SET AS FLAGS
         WHEN AN ARGUMENT)
        2 BYTE PTR TO DATA
        2 BYTE PTR
          (00 00 UNLESS LOCAL AND SAME
           NAME USED AT NEXT HIGHER
           LEVEL IN CALL TREE AND NAME
           IS NOT AN ARGUMENT)
ENTRIES: 1 ENTRY FOR EACH NAME IN SAME
         ORDER AS NAME TABLE
```

Interpretting RLN Statements

With the data tables used by COMAL thus defined, the process of decoding the RLN form of the statements can be continued. Consider again the two COMAL statements shown previously:

```
COMAL STATEMENT
--------------
0010 N:=49152
0020 NUM#:=32767

STMNT'NO LNTH BODY
-------- ---- -------------------------
00 0A    0C   07 01 01 90 40 00 00 00 3A
00 14    09   08 02 02 7F FF 3B
```

Beginning with the first byte of the body of statement 10, the 07 is an operand type code identifying the first operand as a REAL destination. The 01 following indecates the destination is the first entry defined in the NAME table described earlier. The next 01 is an operand type code identifying the second operand as a REAL constant contained in the next five bytes. 90 40 00 00 00 is the number 49152 in standard CBM floating point notation. 3A is the token for the floating point assignment operator (ASGN'REAL). Thus the translated RLN form of the statement is:

```
REAL'DEST   07
N           01
REAL'CONST  01
49152       90 40 00 00 00
ASGN'REAL   3A
```

Operand always begin with a type code followed by a position in the name table or a constant value. The following table summarizes the informaton I have available currently about operands.

OPERAND CHARACTERISTICS TABLE

CODE	OPERAND TYPE	FOLLOWED BY
01	Real constant	5 byte floating pt
02	Integer constant	2 bytes
03	String constant	1 byte length then string
04	Real source	1 byte table position in table
05	Integer source	1 byte table position in table
06	String source	1 byte table position in table
07	Real destination	1 byte table position in table
08	Integer destintn	1 byte table position in table
09	String destinatn	1 byte table position in table

Operator tokens appear to mostly be codes above $10. Those I currently have verified are:

OPERATOR TOKEN VALUES

TOKEN	OPERATOR
$00	REM
$07	STMNT'SEP (;)
$24	MULT'REAL
$27	ADD'REAL
$29	SUB'REAL
$3A	ASGN'REAL
$3B	ASGN'INTEGER
$3C	ASGN'STRING
$3D	ADD'REAL'DEST
$3E	ADD'INTEGER'DEST
$3F	CONCAT'DEST (translates but not implemented)
$40	SUB'REAL'DEST
$41	SUB'INTEGER'DEST
$47	PAREN
$48	ABS function
$59	SQR function
$5F	PRINT'START
$60	PRINT'CRLF
$61	PRINT'END
$64	PRINT'NUMB
$65	PRINT'STRING
$66	PRINT'TAB
$67	PRINT'SPACE
$7F	EXEC
$9C	LABEL
$9D	GOTO
$AA	DATA'START
$AB	DATA'NXT
$AC	DATA'REAL
$AD	DATA'INTEGER
$AE	DATA'STRING
$AF	READ'START
$B0	READ'REAL
$B1	READ'INTEGER
$B2	READ'STRING
$B3	READ'END
$B5	INPUT'START
$B6	INPUT'PROMPT'?
$B7	INPUT'PROMPT'STRING
$B8	INPUT'REAL
$B9	INPUT'INTEGER
$BA	INPUT'STRING
$BB	INPUT'END
$C8	CHANNEL
$CB	CHANNEL'COLON
$D8	PEEK
$9F	END
$A0	STOP

Using the table values it is now easy to inerpret statement 20. The 08 indicates te first operand is an integer destination and is the second entry in the name table. The second operand is an integer constant with a value of $7FFF and finally, the operator is integer assignment.

Thus the translated RLN form of the statment is:

INTEGER'DEST	08
NUM#	02
INTEGER'CONST	02
32767	7FFF
ASGN'INTEGER	3B

LETTER FROM A CARTRIDGE USER

The cartridge is spectacular! It took me only one hour to convert the graphics support package I sent you last month. I will now add a routine for logarithmic axes as well, and send it to you with a demo driver (infrared flux from interstellar dust clouds? I'll try to keep it simple).

My wife is a technical support engineer for a major computer and instrumentation company and has generally treated my C64 with amused tolerance. However, when I got the cartridge last night, I promptly modified DIFFERENTIATE on DEMO DISK #2 to work interactively, and left it on the TV for when she got home. She bit the bait, and differentiated some pretty exotic functions before quitting.

BEST SELLING COMAL BOOKS

Each issue we will repport on the top 5 selling COMAL books. This issue, we get to report on the books for three months (there are advantages to publishing late). The list is pretty much the same each month, except BEGINNING COMAL went out of stock and thus dropped off the list. However, there are now 3 more **NEW** COMAL books out with one more due in January. THE COMAL LIBRARY OF FUNCTIONS AND PROCEDURES is a book and disk set by Kevin Quiggle - perfect for a programmer who doesn't like 'reinventing the wheel'. GRAPHIC PRIMER is another book and disk set, this one from Mindy Skelton - a tutorial on the turtle and sprites in COMAL 0.14. COMAL WORKBOOK should come this month - from Gordon Shigley, author of the now famous TUTORIAL DISK. Finally, Reston should be releasing COMMODORE 64 GRAPHICS WITH COMAL in January. We have had lots of requests for it and as soon as it is out we'll have it (yes, we have 100 copies on order already). With these newcomers perhaps the COMAL books now in the top 5 will have some competition.

SEPTEMBER 1984

#1 - COMAL HANDBOOK
by Len Lindsay
#2 - COMAL FROM A TO Z
by Borge Christensen
#3 - BEGINNING COMAL
by Borge Christensen
#4 - FOUNDATIONS IN COMPUTER STUDIES
by John Kelly
#5 - STRUCTURED PROGRAMMING WITH COMAL
by Roy Atherton

OCTOBER 1984

#1 - COMAL HANDBOOK
by Len Lindsay
#2 - COMAL FROM A TO Z
by Borge Christensen
#3 - FOUNDATIONS IN COMPUTER STUDIES
by John Kelly
#4 - STRUCTURED PROGRAMMING WITH COMAL
by Roy Atherton
#5 - CAPTAIN COMAL GETS ORGANIZED
by Len Lindsay

NOVEMBER 1984

#1 - COMAL HANDBOOK
by Len Lindsay
#2 - COMAL FROM A TO Z
by Borge Christensen
#3 - FOUNDATIONS IN COMPUTER STUDIES
by John Kelly
#4 - STRUCTURED PROGRAMMING WITH COMAL
by Roy Atherton
#5 - CAPTAIN COMAL GETS ORGANIZED
by Len Lindsay

COMAL 0.14 KEYWORD TABLE, DEBUG &

R W Illman sent us the following note: In my exploration of the COMAL KEYWORD TABLE, I find that the table occupies memory from 2068 to 3018 (not 2762 as indicated in your memory map). It contains 177 entries, six of which are null. Interestingly, the dollar sign and double quotes do not appear, but DEBUG and & are included. I wonder what they are intended to do? Is there a story there?

ED NOTE: Yes, the story. Try DEBUG as a direct command. You get a NOT IMPLEMENTED reply. This is the truth. It never was implemented. We knew about DEBUG, but the & was a surprise. So, we checked with UniCOMAL (authors of COMAL 0.14 and 2.0). Yes, & is implemented. The story centers on a discussion about whether or not string concatenation should use the plus sign (+). It was suggested to use the & for string concatenation. And COMAL 0.14 will allow it! Try this:

```
AUTO
0010 DIM A$ OF 9, B$ OF 9
0020 A$="TESTING"
0030 B$="123"
0040 PRINT A$&B$
0050             just hit RETURN here

LIST
0010 DIM A$ OF 9, B$ OF 9
0020 A$:="TESTING"
0030 B$:="123"
0040 PRINT A$+B$
```

COMAL 0.14 is smart. It accepts the suggested use of & and converts it to the standard use of +.

COMAL CAN COMPETE WITH BASIC

by Colin Thompson

I don't think the inventors of COMAL 0.14 ever envisioned how popular it would become. The language was intended to be a 'learning' language, and performs that task better than any other. But.... Americans love to tinker. Just look at the number of BASIC dialects. COMAL, the unassuming, timid little learning language is rapidly becoming the language of choice for many top-notch American programmers. 'Why?' is a question that begs to be answered.

Just to keep this is perspective, remember we are talking about the Commodore 64, a home computer, not a small business PC. Commodore programmers usually write programs for other Commodore owners. That's the essence of Public Domain software. The programs these amateur programmers write range from nearly commercial quality to simple demonstrations. Useful programs find their way into the libraries of user groups and are distributed to club members for the enjoyment of all. BASIC has been the king of 'home brew' software for many reasons, but primarily because it is built into each Commodore. Hence the programs are transportable from the author's computor to yours. Here lies the first roadblock in COMAL's path to the top. To RUN a COMAL program, you must first LOAD the COMAL language. COMAL programs won't RUN under BASIC. Neither will PASCAL, SIMON'S BASIC, LOGO, FORTH, or any other language. Fortunately, a disk loaded version of COMAL is available from most (all?) user group libraries in this country. Anyone can get a copy of the COMAL language. All you have to do is LOAD COMAL and you can start enjoying the programs written by others. That's not so difficult.

Earlier, I described COMAL as 'little'. When the COMAL system is LOADed into your 64, you are left with only 9,902 BYTES FREE. That's far less the BASIC's 37K. How could COMAL possibly compete with BASIC in the BYTES FREE department? The answer is it doesn't have to. Most programs that an amateur programmer (you and me) write are less than 10K in length. If you doubt this, take a look at your own library. Most programs will take up less than 35 blocks on the disk. COMAL is limited to about 10K of workspace, but remember that is 10K of COMAL code, not BASIC. Command for command, COMAL is at least as powerful, but usually far more powerful than the equivalent BASIC command. Therefore if a COMAL program and BASIC program do the same thing, the COMAL program will usually consume much less memory. COMAL offers one more way to expand the limits of the workspace: the CHAIN command. If you have designed a COMAL program that exceeds the 10K limit, you can split up the program into two or more modules on the disk. Each module can replace itself with another by CHAINing in another program module. Parameters may be passed in a variety of ways. This ability will limit a 1541 owner to a total program size of about 170K. I've used the technique extensively.

So far we've seen that anyone can get the COMAL language, it's much more powerful than BASIC, and the size of a COMAL program is limited to the storage capacity of the diskette. What other features would COMAL need to displace BASIC from the top of the mountain? How about a good screen editor? COMAL has the best screen editor ever written for a small computer. Surprise, it's the same one that Commodore BASIC uses. COMAL will look very familiar to BASIC programmers. All the keys do the same thing. COMAL's editor has most of the 'extensions' that have been added to the various BASICs, like RENUMber, PASS "disk string", etc.

COMAL is faster than BASIC. Just how much faster depends on the specific operation. Of course the graphics operations will run a hundred times faster because BASIC doesn't have graphic commands, but in general, COMAL is 2 to 7 times faster than BASIC. I once wrote a program in BASIC that proved too slow. Later, I compiled it in Petspeed and it became just fast enough to be useful. A few months later I rewrote the program in COMAL. It was a little faster that the compiled BASIC version. In general, benchmarks are of little use in evaluating languages, but COMAL beats BASIC everytime. You've all seen BASIC game programs that would be great if they were written in machine language. Those same programs, running under COMAL become playable. I see COMAL as an alternative to machine language for those of us that prefer a high level language.

Any BASIC programmer worth his salt will examine listings of programs written by others. This is how we learn. This is also how we get frustrated. BASIC, in all its unstructured glory, is impossibly difficult to read. COMAL, on the other hand, is structured, making the program legible to others. You can examine the code from a COMAL program and have a very good idea of how the program works. You can even take portions of a COMAL program and add it to another program, easily.

A few people have voiced their dissatifaction with COMAL because it lacks some BASIC commands. True, it doesn't have GET$, ASC, GOSUB, STR$, and VAL . These functions were not included in COMAL 0.14 for reasons known only to the original authors. (GOSUB is not applicable to COMAL). However, COMAL is extendable, something like Forth. You can write your own commands. I do it every day. GET$ and the others have been written and may be entered into any COMAL program that needs them. Actually, almost any command you could design may be written in COMAL as a PROCedure or FUNCtion. This is an exceptionally powerful concept.

Finally, what Commodore BASIC lacks is commands to control music, sprites, and high resolution graphics. COMAL does, almost. I say almost because the disk loaded version 0.14 does not have music commands, but the Professional version 2.0 does. Music commands in 0.14 are loaded as a separate program. COMAL has the highly touted LOGO turtle. This ability opens up a whole new field of programming to frustrated BASIC programmers. Sprites are simple to design and use in a COMAL program.

Taken as a whole, all of the advantages COMAL has to offer the amateur programmer is overwhelming. The new COMALite has a wealth of books, reference manuals and more than 20 disks of COMAL programs to learn with. Many user groups are offering classes in COMAL. National magazines publish articles on COMAL and will soon begin to publish program listings, as well. In short, COMAL is on a roll. COMAL 2.0 will soon be available on the IBM PC and is already being written for 68000 based machines like the MacIntosh. Of course CP/M COMAL has been in use on the Apple for a long time. COMAL runs on the DEC VAX and PDP-11 also. This is not a little-heard of, fly by night language. This is a language you can learn and use. It's the best 'Public Domain' language available to Commodore users.

I haven't told you why so many programmers are using COMAL. I listed several advantages, but the real reason why top-flight programmers like Valerie Kramer, Jesse Knight, Glen Colbert, John McCoy and David Stidolph have switched is very simple. COMAL is fun. Just ask any COMAL programmer, but be prepared to listen for a few hours.

OKIDATA 92 SUPPORT PLUS CHINESE SCREENS

by Craig Van Degrift

Most of these routines are only of specific interest to C64 COMAL users who have an Okidata 92 printer and either a CardCo?/G+ or Turboprint Interface. One routine is a variation of "HI" (on TODAY DISK #5 as OKI92'HI) which lets the user select colors and printer options. I have also included the built in error messages (RAM'ERRORS) option. A second routine performs a HI-RES screen dump to the Okidata 92 in only 20 seconds. It is entirely in machine language and loads into the RS-232 buffer at CE00 to CF3C. The routine is loaded and demonstrated by the program OKI92'SCREEN'IO (on TODAY DISK#5 along with OKI92.DUMP.OBJ).

My subject for the sample screen dump is likely to be of interest to COMALites with an interest in oriental languages. There are two screen fulls of 80 Chinese characters. These are the first 160 characters that the Japanese children must learn in elementary school. I feel that COMAL is an excellent language to make educational programs for the teaching of these characters. Each fits in a sprite and can be stored and used with sprite commands although I have prined them on the high resolution screen to make an interesting screen dump. There are many useful utilities still needed to make playing with these characters easy. I will offer more as they are ready and wish to encourage others to contribute ideas and routines. Certainly the added features of COMAL 2.0 will make this sort of project even easier. Thank you for providing such a great language!

11,838 BYTES FREE IN COMAL 0.14

by John H McCoy

Ed. Note: Here is a program that will give you about 20% more free memory with COMAL 0.14. But, do **NOT** run the program twice. The 2nd time it will relocate the wrong code and crash the system. The program is built into the HI program on TODAY DISK #5. Now here is his letter:

I have ordered the new cartridge and look forward to using it. However, the cost will reduce its attractiveness to many. Thus I am still interested in a disk version which can be freely circulated in public schools... I did teach a graphics course this summer using COMAL and it went very well. Memory was the limiting factor. I have dug into COMAL enough to find that one can get about 900 additional bytes with a simple:

```
POKE 2066,144
POKE 2067,179
NEW
```

To get more room, 13 bytes at $B391 must be moved and the pointers in COMAL adjusted. The total RAM available is then 11,838 bytes. The program EXPAND'COMAL does the trick.

I have not encountered any problems with COMAL using this space. If, however, a program uses a machine language routine and hides it under a DUMMY variable at the start of the proram then this expansion uncovers it. The solution is to write position independent machine code (not always possible) and load at the top of RAM pointed to be 2066-2067. I have incorporated this in an improved version of the PIE'CHART program (it now supports explosion - this program is on TODAY DISK #5).

There is a type in the memory map on page 21 of TODAY#4. Address $2F65 should be $2F53.

Some of the location identifications also need updating. I am probably responsible for the inaccurate description of some locations in the $003A to $0044 area. So I propose the following descriptions as best I know them now:

```
$003A Start of data pointers
$003E Bottom of procedure call stack
$0040 Top of procedure call stack
$0042 Top of local storage stack
$0044 Bottom of local storage stack
      and highest location used by COMAL
```

```
// save "expand'comal"
// comal 0.14 ram expander
// written by john mccoy
//            huntsville, tx
//            october 8,1984
//
// relocate code from $b391-$b39d
//                  to $b791-$b79d
b391:=hex'dec("b391")
b39d:=hex'dec("b39d")
for addr:=b391 to b39d do
 poke addr+1024,peek(addr)
endfor addr
//
// change comal jsr's
//
poke hex'dec("11d9"),183
poke hex'dec("3b98"),183
poke hex'dec("4432"),183
poke hex'dec("4928"),183
poke hex'dec("5549"),183
poke hex'dec("5f0c"),183
poke hex'dec("601e"),183
poke hex'dec("68f2"),183
poke hex'dec("7027"),183
poke hex'dec("7037"),183
poke hex'dec("7519"),183
//
// change comal hi'mem default
//
poke 2066,144
poke 2067,183
print "ram has been expanded to"
print " 11,838 bytes."
print "type new and size to verify"
end
func hex'dec(h$) closed
 l:=len(h$)
 if l=0 then
  return 0
 else
  d:=ord(h$(1:1))-48
  if d>9 then d:=d-7
  for i:=2 to l do
   a:=ord(h$(i:i))-48
   if a>9 then a:=a-7
   d:=16*d+a
  endfor i
  return d
 endif
endfunc hex'dec
```

FUN WITH COMAL 2.0

by Captain COMAL

There are some really nice things built into the COMAL 2.0 CARTRIDGE. For instance, let's have a constant display of the time on the screen, all done as direct commands - no program necessary!

First, we must tell COMAL to USE the SYSTEM package, which has the commands for time:

USE SYSTEM

Now we should set the time:

SETTIME("10:34")

Of course you would set the time properly. In the above example it was set to 10:34 (we left off the seconds - COMAL just uses 00 then). Now we are ready. First clear the screen:

PAGE

And this is it! Here is the entire digital clock command:

WHILE TRUE DO PRINT AT 10,10: GETTIME$

What? Did someone say the numbers were too small so it doesn't qualify? OK, then we will use the built in big letters on the graphics screen. Run this program now (you already set the correct time above):

```
AUTO
0010 USE system
0020 USE graphics
0030 graphicscreen(1)
0040 textstyle(1,2,0,0)
0050 WHILE TRUE DO plottext(50,100,getti
     me$(1:8)) // wrap line
0060               just hit STOP key here
RUN
```

That's it for now. See you next issue.

127 COMAL COLORS

by Eric Edward Haas

Here is a quick little program that shows how you can mix the standard 16 colors to obtain what looks like 127 different colors on your Commodore 64 screen.

```
// delete "0:color mix"
//  by eric edward haas / compucats
// save   "0:color mix"
// august 1984
//
print chr$(147)
border 0
background 0
setgraphic 0
hideturtle
solids
//
for bac#:=0 to 15 do
 for pen#:=0 to 15 do
  pencolor pen#
  background bac#
  if pen#>=bac# then
   checkerboard(pen#*16,bac#*8)
  endif
 endfor pen#
endfor bac#
//
background 0
pencolor 5
plottext 112,192,"press any key"
//
while key$<>chr$(0) do null
while key$=chr$(0) do null
//
pencolor 15
settext
print "press f5 to see the hi-res screen"
end
//
proc checkerboard(x#,y#) closed
 for i#:=x# to 15+x# step 2 do
  for j#:=y# to 7+y# step 2 do
   plot i#,j#
  endfor j#
 endfor i#
 for i#:=1+x# to 15+x# step 2 do
  for j#:=1+y# to 7+y# step 2 do
   plot i#,j#
  endfor j#
 endfor i#
endproc checkerboard
//
proc solids closed
 for n:=0 to 15 do
  background n
  m:=n+1
  if m>15 then m:-15
  pencolor m
  plot 2+n*16,2+n*8
  plot 10+n*16,2+n*8
 endfor n
endproc solids
```

ATTACK OF THE MUTANT COMALS

Reprinted with permission from the newsletter of the Clark County Commodore Computer COMAL Club, issue #4, page 13. Thank you David Skinner!

In the July 1984 issue of COMPUTE!'s Gazette, Richard Mansfield wrote an article entitled 'What is Machine Language' in which he compared machine language with all other languages, "**even mutants like COMAL.**" He went on to say "Why is ML so fast? Because it's the only language which doesn't need some kind of translating before the 6502 chip can understand it." As it turns out, Richard Mansfield may be right about machine language, but what about COMAL? [ED NOTE: most people refer to COMAL as a HYBRID, a change for the better.]

PART 1 - MUTANT COMAL?

MUTANT may be defined as undergoing the act or process of changing. COMAL versions 0.12 for the PET, 0.14 for the C64, 1.02 for the CBM 8096, 2.0 for the C64, and other versions of COMAL for APPLE, BBC MICRO, ACORN, and IBM all prove this description to be accurate. How can we reconcile this with Len Lindsay's claim that COMAL is STANDARDIZED? This is the question we sought to answer at the last club meeting.

Standardization is important because without it, all of our efforts, all of our work can be rendered obsolete. In the case of Commodore BASIC, programs written on the very first PET may well run with little or no change on the C64. There are however may dialects of BASIC used on RADIO-SHACKS and different versions of MS BASIC for 16 bit machines. There is little compatibility between machines.

We will therefore begin our test as one of compatibility. We start by ENTERING a program written in COMAL 0.12 for the CBM 8032 named EIGHTQUEENS.L. The program entered without any problems. When we type RUN, the only difficulty was that the display was formatted for 80 columns. We reduced the leading blanks on lines 280 and 320 and found the program worked perfectly.

It was felt this was not a valid test, it was far too easy. We selected another program written in COMAL 1.02 for the PET 4032 named COMALDICE.L. This program also LOADed without incident. When we typed RUN, we immediately had an error on line 80 CURS$=GET$(0,1). As it happens GET$ is a built in string procedure in COMAL 1.02 and is not implemented in COMAL 0.14. We simply changed line 80 to CURS$=KEY$ and the program worked. [ED NOTE: KEY$ is the way get a character from the keyboard now. Previously, GET$ was used with file 0 meaning keyboard. Other COMALs will also be using KEY$] It still did not look correct. When we listed the program I noted line

```
 6 POKE 59468,12 // GRAPHICS
```

which of course would only work on the PET [ED NOTE: PEEK and POKE are machine dependent]. We changed this line to

```
 6 ;CHR(142) // GRAPHICS
```

which lists as

```
0006 PRINT CHR$(142) // GRAPHICS
```

and then we had a perfect display.

By this time we were beginning to feel rather sure of ourselves. We begin the next test loading a program written in COMAL 1.02 for the CBM 8096 named LABYRINTH 1.02. This time COMAL really surprised us. The disk clattered to a stop and line 860 CURSOR 19,13 was displayed, the message SYNTAX ERROR appeared below the line, the cursor stopped right where we needed parentheses for COMAL 0.14. After enclosing the numbers in brackets, the ENTERing process continued. Every time there was an error of any kind, COMAL would stop, tell us what was wrong and give us a chance to correct it. After the program was loaded, our next job was to ENTER "CURSOR.L" on the TODAY DISK #3. When we type RUN we found one other problem. This one statement: DIM fig$(0:1,0:1,0:4,0:1) OF 400 ended this test. Some quick multiplication showed that this one line allocated 16,000 bytes of memory. We had 5,000 available but we were using a 64K machine; the program was written for a 96K machine. Even though we did not

succeed in getting this program to RUN, we were extremely pleased at the ease with which alien programs could be loaded.

Our next test was the COMAL programs written on an APPLE in John Kelly's book FOUNDATIONS IN COMPUTER STUDIES WITH COMAL. There are only 17 changes to make in the entire book! As an example: on the APPLE we use CLEAR to clear the screen, on the C64 we use PRINT CHR$(147) to clear the text screen, CLEAR is used to clear the GRAPHICS screen. [ED NOTE: The command PAGE is being standardized as the way to clear the text screen with COMAL and is implemented in COMAL 2.0 for the C64 as well as other COMAL implementations].

Our final test was to attempt to load a BASIC program which had been converted to a text file for transmission over a modem. First, make sure the line numbers do not exceed 9999. As each line is loaded you will be given opportunity to correct the errors on that line. Do not attempt to add any other lines at this time. You may leave GOSUB'1000 as the name of a procedure [ED NOTE: notice the added ' apostrophe] and you may leave GOTO L1000 as COMAL can handle GOTO's. You will have to set up a label as follows: 999 LABEL L1000. Sometimes it is better to rewrite a program rather than convert it, but it could save you alot of time if you have a long list of data statements.

The TODAY DISK #3 contained a large number of procedures and functions which are built into COMAL 2.0. It would be a good idea to enter these last and label them so that if you have to convert to 2.0 it will be painless. [ED NOTE: Now even better -- The book / disk set: LIBRARY OF FUNCTIONS AND PROCEDURES by Kevin Quiggle at $19.95]

One of our early conversions is the DATEUNIT.PKG which was copied from PASCAL. There were only a few changes to get it working.

On the question of compatability, we have entered programs from four different machines into the C64, we have entered BASIC and PASCAL text files into COMAL 0.14 with only minimum difficulty. COMAL has certainly proven its ability to be upward compatable. We know that the C64 may not be around forever. I think I would prefer to use COMAL, the language of the future.

ED NOTE: Now that IBM has announced COMAL 2.0 for the IBM PC due about March 1985, you can develop COMAL programs for the IBM PC right now using the C64 COMAL 2.0 Cartridge! The programs should work on the IBM PC with its COMAL 2.0 unchanged for the most part. There is even a cable in the works that will direct connect a C64 to an IBM PC. Then programs can be sent from the C64 direct to the IBM PC. Next issue we may be able to report on this.

COMAL 0.14 - JUST FOR FUN

This little program illustrates an interesting quirk in COMAL. Enter the following program and RUN it:

```
OPEN FILE 3,"XXX",WRITE
PRINT FILE 3: "100 "+CHR$(97)+CHR$(109)+
CHR$(116)+" = 50" // wrap line
PRINT FILE 3: "200 PRINT "+CHR$(97)+CHR$
(109)+CHR$(116) // wrap line
CLOSE
```

After RUNning the program enter the command: NEW. Then switch to UPPER/lower case mode (press SHIFT and the Commodore logo key). Now issue the command:

```
ENTER "XXX"
```

Now LIST the program in memory. Observe that the keywords are lower case, variables names are UPPER CASE. Also note that you can RUN it and LIST it, but you cannot change either line.

The quirk is that COMAL is designed to accept the ASCII codes. In ASCII, codes 65 thru 90 represents A-Z while 97 thru 122 represent a-z. Unfortunately Commodore uses a modified ASCII and shifted alphabetic characters are coded 97 thru 122. However, COMAL rejects all codes above 127 as invalid. Thus a program which uses the codes 97 thru 122 can be ENTERed from a file but can't then be edited from the keyboard.

STRINGS IN COMAL

by Len Lindsay

Several people have now written asking about the missing MID$, LEFT$ and RIGHT$ in COMAL. Actually, these are not missing, they are just implemented differently (better after you understand the COMAL way). Appendix B of the COMAL HANDBOOK explains COMAL string handling. This article will only cover some important aspects.

What MID$, LEFT$ and RIGHT$ are actually doing is providing your program with a SUBSTRING. In COMAL, you chose the substring by naming the string followed by the characters needed specified inside parentheses:

```
A$(<start character> : <end character>)
```

If DATE$="12/25/84" we can find the month, day, and year by using substrings:

```
PRINT date$(1:2) // print month
PRINT date$(4:5) // print day
PRINT date$(7:8) // print year
```

In Microsoft (Commodore) BASIC this would look more contrived:

```
PRINT MID$(D$,1,2):REM PRINT MONTH
PRINT MID$(D$,4,5):REM PRINT DAY
PRINT MID$(D$,7,8):REM PRINT YEAR
```

Of course you can use variables to chose the start and end character positions. Both start and end positions are optional. For example, to specify just the third character in the string:

```
PRINT date$(3)
```

If you want the first five characters, simply use 1 as the start:

```
PRINT date$(1:5)
```

But the best part is that you can change any part of a string without affecting the rest of it (not possible in BASIC). For instance to change the third character from "/" into "x" we would simply type:

```
date$(3):="x"
```

Now, date$ is equal to "12x25/84".

This same string handling applies to string arrays too. Just specify the substring points right after the array index:

```
PRINT NAME$(5)(1:6)
```

This would print the first 6 characters of the 5 element of NAME$ array. It is all consistent, so you also can change any part of any element of a string array.

I'll conclude with one subtle point. What if you specify a range of characters to be changed, but don't assign enough characters to it? Well, COMAL will just fil in spaces to get to the end character. For example:

```
A$:="ABCDEFG"
A$(3:5):="X"
PRINT A$
ABCX  G
```

Use of this feature may not be apparent, but it does come in handy. For instance, you pad spaces on to the end of a string to make it exactly 20 characters long easily now:

```
NAME$(1:20):=NAME$
```

Did you catch that? We told COMAL to assign characters 1 thru 20 of NAME$ to be equal to NAME$. Now if NAME$ starts out as "BOB", COMAL will add spaces to it until the 20th character. It also is easy to blank out a string variable now by turning it into all spaces rather than just a null string:

```
NAME$(1:20):=""
```

For more information on strings, see the COMAL HANDBOOK appendix B.

BUG IN COMAL TODAY #2 PAGE 29

There was a bug in the program listed on page 29. Procedure LP'CONDENSE needs a quick change:

```
OLD: OPEN FILE 13,"",UNIT 4,8,WRITE
NEW: OPEN FILE 13,"",UNIT 4,13,WRITE
```

YOU'RE LOOKING GOOD!

or

Unlearning Some BASIC Ideas

(Part 2)

by Steve Kortendick

This is the second article in a series comparing COMAL to the programming language BASIC, with an emphasis on the effects they have on the programmer. For those of you who missed the first article, and to refresh the memories of those of you who read it weeks ago, Part 1 set the scene by arguing that we are affected by our programming. Our relationships together contribute toward the way we are; in particular, our use of language provides the framework in which we think. The intent of this series is not to detail how each of us is changed in our relationship with the computer; that's a task I leave to the behavioral scientists. The intent, rather, is to demonstrate that, whatever the changes to us may be, those introduced by BASIC are inferior and even counterproductive, while those encouraged by COMAL are far more constructive, enlightening and helpful.

Although some of the strongest points in this argument are in the area of program construction and instruction flow, I resist the tempation to play the trumps and will open with the low suits, the often overlooked aspects of any language. We begin with one of the first differences the COMAL novice notices compared to his or her BASIC experience: program line entry.

Both languages were designed to be uncompiled and interactive, so they are usually integrated with program editors, interpreters or semi-compilers, and run-time modules "built in." In fact, many users of these languages are unaware that most languages require a separate word processor-like progam editor, a compiler, and a linker. But even the editors differ between the two sets of environments.

The first difference one might notice is that the majority of BASIC editors don't even have a command to help number lines for program entry (with IBM PC BASIC a clear exception); most COMALs contain AUTO in their command set. This allows a programmer to enter lines unconcerned about typing (or mistyping) line numbers as they are entered. Experience has taught me, too, that entering 1450 instead of 1540 is an awfully common error. It has the double consequence of leaving nothing for line 1540 and obliterating the previous contents of 1450. The Commodore 64's 2.0 COMAL cartridge begins AUTO numbering after the last current line number, or, if the number's default value is overridden with a parameter, even highlights line numbers which already exist as a warning that entering the new line could cause an error.

Few personal computer implementations of BASIC validate each line as it is entered. Keying errors are not caught until the interpreter attempts to parse an unintelligible line; then the ubiquitous

```
SYNTAX ERROR IN LINE 2455
```

sounds its familiar tune, stops program execution, and, if the error is repaired, marks the program unfit to CONTinue. COMAL, on the other hand, parses each line as it is entered. If it does not understand a line, it places the cursor on the offending character, and offers an explanation of the error. COMAL 2.0 is quite thorough in its suggested corrections, with such comments as:

```
 numeric expression missing
            and
 ":=" or "(" expected, not name
```

These are even available in English, Danish, or (using a disk-loaded package) French (other languages will be available soon).

Even COMAL, of course, can't catch every error. If I entered the line:

```
 25 primt
```

COMAL would accept it, not knowing I wanted **"25 print"**. If, on the other hand, I entered:

```
25 primt variable$
```

COMAL would say:

":=" or "(" expected, not string name

with the cursor positioned on the v of variable$. This is COMAL's way of telling me that it thinks "primt" is a procedure or function which will be defined later, or a variable name, rather than a way to misspell "**print.**"

A valuable side-effect of COMAL's storage of parsed, "understood" lines is that it "forgets" how they were actually keyed. COMAL concentrates on the meaning of the line, and displays it the same way each time. That means that, except within REMarks, a consistent number of spaces is included in each listing, regardless of how the line was keyed. For example, if I changed "COS(angle*2)" to "COS(angle)" by blanking out the "*2", COMAL would close the gap on subsequent listings; BASIC would leave it alone. Another feature is that COMAL makes all line numbers exactly four digits long, giving a more tailored look to the listings. All this makes LISTings easier to follow and to understand.

The next aspect of COMAL likely to be noticed by the novice is the limitation of one command per line. Although BASIC allows

```
10 CO=COS(AN):PRINT CO:AN=AN+PI/4:REM PR
INT THE COSINE AND INCREASE THE ANGLE
```

COMAL insists upon

```
0010 cosine=COS(angle)
0020 PRINT cosine
0030 angle:+PI/4 //increase the angle
```

Forcing us programmers to be keep the lines discrete simplifies debugging, allows insertions to be made into the commands very easily, clarifies the intent of each section, and aids in documentation of the program.

Another visual aspect of COMAL 2.0 listings is the very simple but useful fact that COMAL LISTs keywords and variables in different cases, usually keywords in upper case. In the one of the examples above, "primt" would be listed "primt", while "print" would come out "PRINT".

Our final listing convention is probably the most useful. This is the automatic indentation of subordinate program lines. Never again is there any question about where a FOR loop ends or what the range of an IF should be. The difference is obvious:

```
0100 IF score=100 THEN
0110   PRINT "PERFECT Paper!"
0120   mark'finished
0130 ELSE
0140   PRINT "Room to improve"
0150   try'again
0160 ENDIF
0170 enterscore(score)
```

versus

```
90 IFSC<100THENPRINT"PERFECT
Paper!":GOSUB500:GOTO110
100 PRINT"Room to improve":GOSUB600
110 GOSUB900
```

The verdict is yours.

All of these advantages may seem to affect only the aesthetics of COMAL over BASIC. In fact, they do affect your programming, both in accuracy and personal appreciation. Program code should not be subconsciously understood as random symbols on the screen or paper; they are organized, highly-structured instructions to the machine. The more they appear that way, the better the programmer will understand his or her role and relationship to both the machine and the task at hand.

The next issue will feature a discussion of what appear to be some of the more substantive issues. That understanding, however, belies the importance of the way language affects our understanding.

INTERRUPT INPUT - EASY RESTART

William Wood sent us this note: Please do not forget to mention that you can even NEW a disk or CAT a disk or LIST a program while stopped (hit STOP key) at an INPUT statement and still repeat that input statement when the command: CON is given. I sometimes get ahead of myself and find the ability to STOP and then repeat an input statement extremely convenient.

COMAL 0.14 MACHINE CODE ROUTINES

By David Stidolph & Jesse Knight, Jr.

A program ("ML'SETUP") is on Today Disk #5 that adds several capabilities to COMAL 0.14. This program puts extra machine code routines into the RS-232 modem buffer. Once they are in this buffer, you can LOAD, SAVE, CHAIN, ENTER, etc without affecting them (but don't use the modem buffer). The following routines are included:

SPRITE FLIPPING

Hi-Res or Multi-Color sprites can be flipped upside down or mirrored (flipped from left to right). The sprite mirror machine language program was written by Jesse Knight, Jr. Using the procedures in "ml'procs" you can get four different images from one sprite definition. The commands are:

```
spriteflip'left'right(block)
spritefilp'up'down(block)
```

Block is the sprite image definition you want flipped (images number 0-55).

AUTOMATIC CLOCK

This program runs off the built in IRQ interrupt structure of the C64. Every 60th of a second the 64 stops what it is doing and updates the Jiffy clock and checks the keyboard. This program takes the time from the Time of Day Clock (TOD) and puts it in the top right hand corner of the screen at every interrupt. The time is displayed in whatever color the pencolor is. Since the TOD clock is used, disk access does not slow it down (like it does the Jiffy clock).

Turn clock on with: display'clock(true)
Turn clock off with: display'clock(false)

READ'SPRITE

This routine allows you, with the procedures in "ml'procs", to define sprite images with strings of characters instead of numbers. There are seperate routines for Multi-Color sprites and Hi-Res sprites. To define a Hi-Res sprite you use a "." to indicate a transparent pixel and a "#" to indicate it is on. The following is an example set of data statements.

```
1000 data ".......######......."
1010 data "......###..###......"
1020 data ".....###....###....."
1030 data ".....###....###....."
1040 data ".....###....###....."
1050 data "......###..###......"
1060 data ".......######......."
1070 data "h"
```

You will notice that the each line of the string is less than 24 characacters and there are far less the the 21 lines of a sprite. The readsprite procedures will assume the rest of the sprite is not on. The last line of a sprite (even if you define all 21 lines) should be "h" to indicate the end of the Hi-Res sprite. To read the sprite image from data statements requires the procedures in ml'proc, and to use the procedure "readsprite'h(block'number)". This will read in the Hi-Res sprite and put it in the proper sprite block area (0-55).

Multi-Color sprites can display 3 colors besides the background (or transparent). It can do this by using pairs of pixels to define the four possible colors (this means you have only 12 pixels across, but each dot is double width so the apparent width is the same as a Hi-Res sprite). The colors are represented by:

```
0 = Transparent
    (see through sprite to background)
1 = Sprite Multi-Color Register #1
    (common to all sprites)
2 = Sprite Color
    (specific to each sprite)
3 = Sprite Multi-Color Register #2
    (common to all sprites)
```

To define multi-color sprites, the following data format is used:

```
1000 data "333333111111" // looks like
1010 data "333333222222" // an American
1020 data "333333111111" // flag.
1030 data "222222222222"
1040 data "111111111111"
1050 data "222222222222"
1060 data "111111111111"
1070 data "m"
1080 //
```

```
1090 setgraphic 0
1100 readsprite'm(0) // sprite block 0
1110 spritepos(1,160,100)
1120 spritecolor 1,1
1130 spriteback 2,6
1140 identify 1,0
1150 end
```

Again, if the data lines are shorter than the normal sprite definition, or fewer lines are present, those areas not defined are assumed transparent, or off. To read in this sprite you would use the procedure: readsprite'm(block'number)

QUOTE MODE

When a quote (") is typed, or the insert key is pressed, pressing a color key, cursor key, or any other control key, results in a graphic character. Many times (like when editing or during a running program) you do not want those graphic characters printed. This machine code program works like the clock program in that it is called every 60th of a second. If the quote'mode is set, then the program will zero the quote and insert registers. The procedures in "ml'procs" control this mode by the following commands:

quote'mode(true) - turns quote mode on
quote'mode(false) - turns quote mode off

Note: The 2.0 cartridge has this command built into the SYSTEM package.

REVERSE GRAPHICS SCREEN

If you are doing graphics on the Hi-Res screen, this machine language program will reverse (any pixel on is turned off and all pixels off are turned on) the entire graphics screen. The command to do this is: reverse'bitmap

If this is tried on the multi-color screen, the colors are changed as follows (normally this would look very strange):

0 becomes 3
1 becomes 2
2 becomes 1
3 becomes 3

SAVING THE GRAPHICS SCREEN

Issue #3 of COMAL TODAY included a procedure to save the graphics screen bitmap. The bitmap tells what pixels are on or off. The color map is not saved, so any image will be two color (whatever the loadscreen procedure sets it up as). That savescreen procedure has a bug (it saves past the end of the screen), but even fixed it takes over 3 minutes to save a picture. This new routine saves the screen in mere seconds.

```
// delete "0:ml'setup"
//  by david stidolph & jesse knight jr.
// save   "0:ml'setup"
//
restore
ml'data
poke 182,0
poke 169,0
//
repeat
 input "enter hour (1-12): ": h
until h>0 and h<13 and h=int(h)
//
print
repeat
 input "enter minute (0-59): ": m
until m>-1 and m<60 and m=int(m)
print
//
repeat
 input "enter seconds (0-59): ": s
until s>-1 and s<60 and s=int(s)
print
//
settime(h,m,s)
poke 182,1 // turn clock on
poke 169,1 // turn quote mode off
//
print chr$(147)
print "to turn clock off - poke 182,0"
print "to turn clock on  - poke 182,1"
print
print "turn quote mode on - poke 169,0"
print "turn quote mode off- poke 169,1"
print
print "to flip sprite left-right"
print "    poke 247,block number"
print "    sys 52736"
print
print "to flip sprite up-down"
print "    poke 247,block number"
print "    sys 53097"
print
```

```
print "to withdraw clock and quote'mode"
print "from system (return to normal):"
print "    sys 52868"
if peek(788)=85 and peek(789)=45 then
 sys 52843
endif
print "reactivate clock & quote'mode:"
print "    sys 52843"
end
//
proc settime(hrs,mnt,sec) closed
 cia1:=56320; ccrb:=56335
 poke ccrb,peek(ccrb) mod 128
 x:=0
 if hrs>12 then x:=128; hrs:-12
 if hrs=0 then hrs:=12
 x:+(hrs div 10)*16+(hrs mod 10)
 poke cia1+11,x
 poke cia1+10,(mnt div 10)*16+(mnt mod 1
 0) // wrap line
 poke cia1+9,(sec div 10)*16+(sec mod 10)
 poke cia1+8,0
endproc settime
//
proc ml'data
 total:=0
 for x:=52736 to 53238 do
  read a
  total:+a
  poke x,a
 endfor x
 //
 if total<>70520 then
  print "error in data statements"
  close
  stop
 endif
 //
 data 165,247,133,249,169,0,133,250
 data 162,6,24,38,249,38,250,202
 data 208,248,165,250,9,192,133,250
 data 160,63,177,249,208,24,160,62
 data 134,248,177,249,162,8,106,38
 data 248,202,208,250,165,248,145,249
 data 136,16,237,24,144,27,160,62
 data 134,248,177,249,162,4,24,106
 data 106,38,248,42,38,248,74,202
 data 208,244,165,248,145,249,136,16
 data 231,160,60,177,249,170,200,200
 data 177,249,136,136,145,249,138,200
 data 200,145,249,136,136,136,136,136
 data 16,233,96,120,173,20,3,141
 data 254,206,173,21,3,141,255,206
 data 169,147,141,20,3,169,206,141
 data 21,3,88,96,120,173,254,206
 data 141,20,3,173,255,206,141,21
 data 3,88,96,141,251,206,142,252
 data 206,140,253,206,162,2,160,0
 data 165,169,240,4,132,212,132,216
 data 165,182,240,53,169,33,153,30
 data 4,200,173,11,220,41,16,76
 data 189,206,189,9,220,32,237,206
 data 189,9,220,32,241,206,169,58
 data 153,30,4,200,202,16,235,136
 data 169,33,153,30,4,173,134,2
 data 153,30,216,136,16,250,173,8
 data 220,173,251,206,174,252,206,172
 data 253,206,108,254,206,74,74,74
 data 74,41,15,24,105,48,153,30
 data 4,200,96,0,0,0,0,0
 data 169,0,168,170,141,50,207,141
 data 51,207,141,52,207,169,128,141
 data 49,207,189,0,4,41,1,240
 data 10,185,50,207,24,109,49,207
 data 153,50,207,232,224,24,240,8
 data 78,49,207,144,229,200,208,221
 data 96,0,0,0,0,169,224,133
 data 251,120,165,1,41,253,133,1
 data 162,31,160,0,132,250,177,250
 data 73,255,145,250,200,208,247,230
 data 251,202,208,242,160,0,177,250
 data 73,255,145,250,200,192,65,208
 data 245,165,1,9,2,133,1,88
 data 96,165,247,133,249,169,0,133
 data 250,162,6,24,38,249,38,250
 data 202,208,248,165,250,9,192,133
 data 250,169,0,141,153,2,169,62
 data 141,154,2,172,153,2,177,249
 data 72,172,154,2,177,249,172,153
 data 2,145,249,104,172,154,2,145
 data 249,206,154,2,238,153,2,173
 data 153,2,201,32,208,221,76,81
 data 206,166,247,32,201,255,169,0
 data 32,210,255,169,224,32,210,255
 data 169,224,133,250,160,0,132,249
 data 162,31,32,225,207,200,208,250
 data 230,250,202,208,245,32,225,207
 data 200,192,64,208,248,32,204,255
 data 96,165,1,72,41,240,120,133
 data 1,177,249,133,248,104,133,1
 data 88,165,248,32,210,255,96
endproc ml'data
```

PART 2 - Next Page

COMAL Cartridge Built in Word Processor

Valerie Kramer sent us this letter:

I don't know if it has occurred to you but we now have a mini-word processor in the COMAL 2.0 Cartridge. No program is required. Just fill the screen with your letter (as I am doing with this one) then hit control-P to print the text screen. Naturally this isn't the most advanced word processor around, but it certainly is convenient.

```
// delete "0:ml'procs"
//  by david stidolph & jesse knight jr.
// list   "0:ml'procs"
//
// before you use these procedures
// you must have run ml'setup or
// proc ml'data in that program.
// otherwise you can crash comal!
//
proc quote'mode(on'off) closed
 if on'off then
  poke 169,0 // funny graphic chars
 else
  poke 169,1 // no funny graphic chars
 endif
endproc quote'mode
//
proc clock'display(on'off) closed
 if on'off then
  poke 182,1
 else
  poke 182,0
 endif
endproc clock'display
//
proc sprite'flip'left'right(blk)
 if peek(52736)=165 then
  poke 247,blk
  sys 52736
 endif
endproc sprite'flip'left'right
//
proc sprite'flip'up'down(blk)
 if peek(53097)=165 then
  poke 247,blk
  sys 52736
 endif
endproc sprite'flip'up'down
//
proc setup'irq//for clock & quote'mode
 if peek(52843)=120 and peek(788)=85 and
 peek(789)=45 then // wrap line
  sys 52843
 endif
endproc setup'irq
//
proc restore'irq  //return to normal
 if peek(52868)=120 and peek(788)=147 an
 d peek(789)=206 then // wrap line
  sys 52868
 endif
endproc restore'irq
//
proc reverse'screen
 if peek(53045)=169 then
  sys 53045
 endif
endproc reverse'screen
proc save'screen(filename$) closed
 if peek(53169)=166 then
  open file 79,filename$+",p",write
  poke 247,79
  sys 53169
  close file 79
 endif
endproc save'screen
//
//the rest of these procedures are
//for reading sprites defined by
//strings.  (i.e. ".....#####...")
//if you don't need them then
//del 9355-   to save memory.
//
func find'string closed
 pointer1:=peek(51)
 pointer2:=peek(52)
 address:=pointer2*256+pointer1
 return address+4
endfunc find'string
//
proc read'sprite'h(blk) closed
 check'sprite'ml'data
 dim line$ of 24
 dim sprite$ of 64
 sprite$:=""; count:=1
 read line$
 while line$(1)<>"m" do
  while len(line$)<24 do line$=line$+"0"
  def'line(line$,sprite$,count)
  read line$
 endwhile
 end'sprite(blk,line$,sprite$,count)
endproc read'sprite'h
//
proc read'sprite'm(blk) closed
 dim sprite$ of 64, line$ of 12
 dim f'line$ of 24
 sprite$:=""; line$:=""
 check'sprite'ml'data
 count:=1
 read line$
 while line$(1) in "0123" and count<64
  while len(line$)<12 do line$=line$+"0"
  f'line$:=""
  for lp1:=1 to 12 do
   case line$(lp1) of
   when "0"
    f'line$:=f'line$+"00"
   when "1"
    f'line$:=f'line$+"01"
   when "2"
    f'line$:=f'line$+"10"
   when "3"
    f'line$:=f'line$+"11"
   otherwise
    print "sprite data statements error"
```

```
    stop
   endcase
  endfor lp1
  def'line(f'line$,sprite$,count)
  read line$
 endwhile
 end'sprite(blk,line$,sprite$,count)
endproc read'sprite'm
//
proc def'line(ref line$,ref sprite$,ref
count) closed // wrap line
 if line$="" then null
 addr:=find'string
 poke 53011,addr mod 256
 poke 53012,addr div 256
 sys 52992
 for x:=0 to 2 do
  sprite$(count):=chr$(peek(680+x))
  count:+1
 endfor x
endproc def'line
//
proc end'sprite(blk,line$,ref sprite$,re
f count) closed // wrap line
 while count<64 do
  sprite$(count):=chr$(0)
  count:+1
 endwhile
 case line$(1) of
 when "h","H"
  sprite$(64):=chr$(0)
 when "m","M"
  sprite$(64):=chr$(1)
 otherwise
  print "sprite data statements error"
  stop
 endcase
 define blk,sprite$
endproc end'sprite
//
proc check'sprite'ml'data
 if peek(52992)<>169 then
  print "error-machine language package"
  print "      not present. please load"
  print "      before running."
  stop
 endif
endproc check'sprite'ml'data
```

TODAY DISK #5 - LOTS OF PROGRAMS

There are alot of programs on TODAY DISK #5 that we don't have room to talk about here. Some are quite good. See the directory listing in this issue. For example, MAGIC FRUIT is a casino game, and EXPLODED'PIE will produce a nice pie chart - with sections slid out a bit. Plus lots of nice graphics.

COMAL 2.0 MODEM USE

By David Stidolph

Comal 2.0 is a marvelous language, but occasional errors do creep in. In Denmark, modems are available only from the telephone company (EXPENSIVE!!!), so the developers couldn't test the modem commands. When COMAL uses GET$ to get characters from the RS-232 buffer, it uses the CHRIN Kernal routine instead of GETIN. This may not make much sense to you, but what happens is that by using CHRIN, the system will wait for a carridge return. The following routine changes the vector for CHRIN to point to GETIN, gets the character (chr$(0) if none) and returns the character. Note that the stop key is trapped. This is because if the routine stops while CHRIN points to GETIN, your computer would crash, and the only way back would be to turn the computer off and back on.

```
FUNC modem'get$(file'number) CLOSED
  old'lo:=PEEK($0324)
  old'hi:=PEEK($0325)
  DIM c$ OF 1
  TRAP
    TRAP ESC-
    POKE $0324,PEEK($032a)//change CHRIN
    POKE $0325,PEEK($032b)// to GETIN
    c$:=GET$(file'number,1)
    POKE $0324,old'lo // restore CHRIN
    POKE $0325,old'hi
    TRAP ESC+
  HANDLER // error in GET$ comes here
    POKE $0324,old'lo // restore CHRIN
    POKE $0325,old'hi
    TRAP ESC+
    REPORT//don't handle the error here
  ENDTRAP
ENDFUNC modem'get$
```

This function could be used like this:

```
modem'line=22
OPEN FILE modem'line,"sp:"
REPEAT
  c$=modem'get$(modem'line)
  print c$,
UNTIL c$="@" // @ received means stop
CLOSE
```

The other features about modem control documented in The COMAL Reference Handbook are accurate, and make modem control easy.

TARGET GAME - A START
by Len Lindsay

It is quite easy to set up nice games using COMAL sprites. I have put together a quick little program that is the beginning of a shoot the targets game. A small ball rolls across the bottom of the screen. The targets are at the top. When you wish to "shoot" simply hit the space bar. The ball then pops straight up to the top. The program detects collisions between the ball and the targets (both are sprites). The program is taken from the book COMMODORE 64 GRAPHICS WITH COMAL with permission. For COMAL 2.0 add a USE SPRITES and USE GRAPHICS at the start of the program and add parentheses around the sprite and graphics parameters.

```
// delete "0:spritecollisn16"
//  (c)1984 lindsay
// save   "0:spritecollisn18"
//
// use graphics  for 2.0
// use sprites   for 2.0
init
for trys:=1 to 3 do game
pencolor 12
settext // textscreen  in 2.0
//
proc init
 setgraphic 0 //graphicsscreen(0) in 2.0
 plottext 1,1,"hit space bar to shoot"
 sprite'box(1)
 for x:=0 to 6 do
  identify x,1
  spritesize x,true,true
  spritecolor x,x
  spritepos x,x*6*8,194
  // showsprite(x)  in 2.0
 endfor x
 spritecolor 6,0
 sprite'dot(7)
 identify 7,7
 spritesize 7,true,true
 spritecolor 7,1
 spritepos 7,1,1
 // showsprite(7)  in 2.0
 moveto 0,199
 pencolor 7
 drawto 319,199
 moveto 0,195
 drawto 319,195
 fill 1,196
endproc init
//
proc game
 y:=10
 for x:=1 to 319 step 2 do
  spritepos 7,x,y
  if key$=" " then
   dummy:=spritecollision(7,true)
   repeat
    y:+3
    spritepos 7,x,y
   until spritecollision(7,true)
   y:=10
   for temp:=1 to 5 do
    if spritecollision(temp,false) the
    n spritecolor temp,0 //wrap line
   endfor temp
   while key$>chr$(0) do null//clearbufr
  endif
 endfor x
endproc game
//
proc sprite'dot(num) closed
 dim sprite$ of 64
 for x:=1 to 64 do sprite$(x):=chr$(0)
 sprite$(1):=chr$(192)
 sprite$(4):=chr$(192)
 define num,sprite$
endproc sprite'dot
//
proc sprite'box(num) closed
 dim sprite$ of 64
 for x:=1 to 63 do sprite$:=sprite$+chr$
 (255) // wrap line
 sprite$:=sprite$+chr$(0)
 define num,sprite$
endproc sprite'box
```

SHOW STOPPER - CAT PAUSER
by Ray Carter

If you're at all like me, one of your pet peeves about disk-based COMAL is that we can't pause a CATalog listing like we can a program listing (I understand that's been taken care of in the cartridge version). The program SHOW-STOPPER handles that problem and will also cause running programs to pause by hitting <CTRL> S. The CAT or program resumes when you hit <CTRL> Q -- much in the fashion of terminals on many larger systems. The program works by entering a machine language routine which intercepts the usual IRQ interrupt (that's what refreshes the screen and also updates the jiffy clock). The program uses the area of free memory between 679 and 743 which doesn't usually interfere with other stuff. PROGRAM ON NEXT PAGE

```
// delete "0:show-stopper"
//  by Ray Carter
// save   "0:show-stopper"
// Southern New Mexico Commodore Users
//
// inserts an irq interrupt which will
// pause a program or 'cat' whenever
//  <ctrl> s is typed
// resumes when <ctrl> q is typed
//
for i:=679 to 743 do
 read j
 poke i,j
endfor i
sys 679
data 120,169,180,141,20,3,169
data 2,141,21,3,88,96,173
data 141,2,201,4,208,42,165
data 197,201,13,208,36,120,169
data 85,141,20,3,169,45,141
data 21,3,88,173,141,2,201
data 4,208,249,165,197,201,62
data 208,250,120,169,180,141,20
data 3,169,2,141,21,3,76
data 85,45,0,0,0,0,0
```

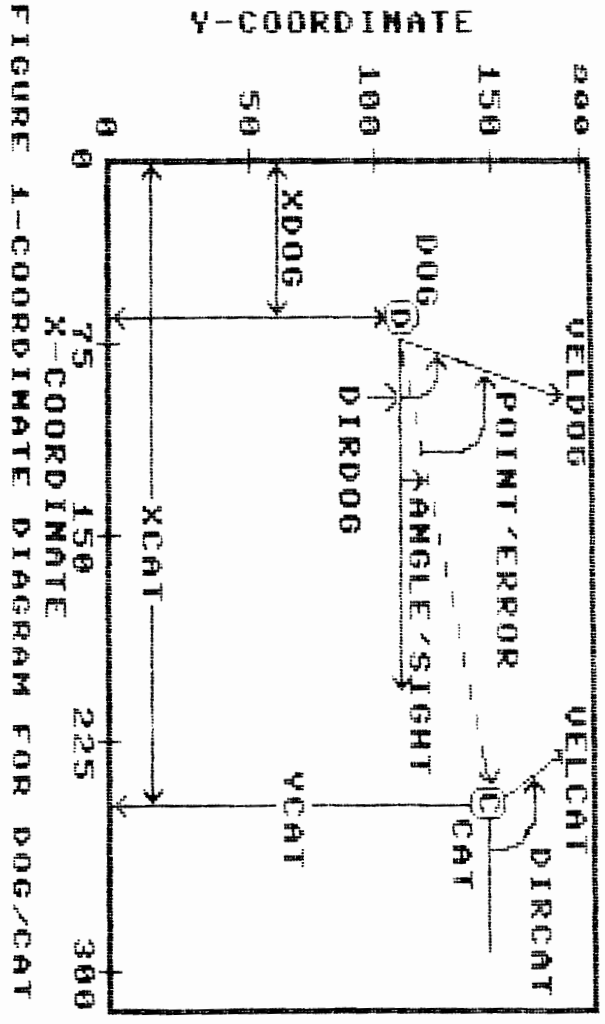


FIGURE 1--COORDINATE DIAGRAM FOR DOG/CAT

DOG CHASES CAT

By Bert Denaci

This game program was written for my young grandsons. It is an example of classical pursuit homing such as a "Dog chasing a Cat". The program uses hi-res graphics to show the paths, and sprites to show the "Dog" and the "Cat". A joystick in port 2 controls the direction of the cat, in an attempt to avoid the dog. If the dog misses the cat, it goes ape and circles around trying to find the cat again. Hint: turn the cat's path 90 degrees to the dog's path to cause a miss.

The problem is illustrated by the coordinate system shown in the accompanying figure, and the following definitions:

XDOG, YDOG - Coordinates of dog
XCAT, YCAT - Coordinates of cat

VELDOG - Velocity of dog
VELCAT - Velocity of cat

DIRDOG - Angular direction of dog
DIRCAT - Angular direction of cat

RATE'DIRDOG - Turning rate of dog and cat
RATE'DIRCAT - Turning rate of cat

ANGLE'SIGHT - Angle of line-of-sight between dog and cat

POINT'ERROR - Pointing error between dog velocity and line-of-sight

T - Time
DT - Time increment between iterations

TAU - Time constant between command turning rate of dog (RATE1'DIRDOG) and actual turning rate

K - Dog turning gain

The program is divided into three parts:

(1) Initial conditions, constants, and setup of graphics and sprites (up to line 360).
(2) Repetative iterative calculations, which use a simple rectangular integration. The calculations continue until the dog catches the cat, or the cat

escapes, or time, T is greater than 9 (lines 380 to 710).
(3) Procedures for quitting the program and for reading the sprites.

Most of the equations may be understood from study of Figure 1. Some extra notes are as follows: Lines 250 and 260 provides the guidance or command turning rate for the dog. Lines 410 and 420 are an algorithim for the first order lag with time constant, TAU, between commanded and actual turn rate of the dog. Lines 480 to 520 allow a joystick in prot 2 to command a turning rate for the cat. Line 620 stops the run if the dog catches the cat, and lines 630 to 660 stops the run if the cat reaches the border. Finally, lines 800 to 1010 contains the procedure for reading the sprites. This procedure is a modification of a corresponding procedure contained in the program "SKY'CATCHER", published in POWER/PLAY, June/July 1984.

If you like to experiment, try changing K and/or TAU to make it easier or harder for the cat to avoid the dog. Changing the dog and cat velocities, maximum turning rates, and range to quit (line 620), can also make it easier or harder for the cat to escape.

SIGN LANGUAGE
by Carl Frerichs

COMAL is now branching out. It is proving to be quite an all purpose language. Carl Frerichs sent in this program, which provides a means of practicing sign language. And the program is good too! I had my seven year old daughter, Rhianon, try it out. She already knows a bit of sign language, and could identify the letters correctly when they were displayed on the screen. This program relies on data on the disk, so it can't be LISTed here. But, even if you are not interested in sign language, you may find the program itself quite interesting. It is a small but jazzed up program. There is no memory problem as the information the program uses is on disk and used as needed. Also, future expansion does not depend on memory size but disk space. And there is a lot of that. Now, shall we start collecting data files that will work with this system? Send them in!

SHOW YOUR OWN TURTLE
by David Stidolph

If you want an image, other than the triangle shape called a "turtle", to move around the screen while it draws, you can IDENTIFY any sprite image for sprite number 7 and the computer will continue to show it as if it were the turtle. The sprite will not be rotated by LEFT and RIGHT, but you can expand the image with SPRITESIZE. In COMAL 2.0 remember to issue a USE SPRITES command first. A program to demonstrate this is on TODAY DISK #5. For instance, if sprite image number 30 has a definition of a colored turtle, and you want to use it as the new turtle image, you would use the following statements:

COMAL 0.14	COMAL 2.0 Cartridge
IDENTIFY 7,30	IDENTIFY(7,30)

```
// delete "0:turtle/demo3"
//  by david stidolph
// save   "turtle/demo5"
//
func find'string closed
 pointer1:=peek(51)
 pointer2:=peek(52)
 address:=pointer2*256+pointer1
 return address+4
endfunc find'string
//
proc read'sprite'ml'data closed
 //
 data 0,0,0,169,0,168,170,141
 data 168,2,141,169,2,141,170
 data 2,169,128,141,220,2,189
 data 0,4,41,1,240,10,185,168
 data 2,24,109,220,2,153,168
 data 2,232,224,24,240,8,78,220
 data 2,144,229,200,208,221,96
 total:=0
 //
 for x:=680 to 731 do
  read a
  poke x,a
  total:+a
 endfor x
 //
 if total<>5747 then
  print "error in data statements"
  stop
 endif
endproc read'sprite'ml'data
//
```

```
proc read'sprite(blk) closed
 if peek(683)<>169 then
  read'sprite'ml'data
 endif
 dim line$ of 24
 dim sprite$ of 64
 sprite$:=""; count:=1
 read line$
 while (line$(1)="0" or line$(1)="1") an
 d count<64 do // wrap line
  while len(line$)<24 do line$=line$+"0"
  if line$="" then null
  addr:=find'string
  poke 702,addr mod 256
  poke 703,addr div 256
  sys 683
  for x:=0 to 2 do
   sprite$(count):=chr$(peek(680+x))
   count:+1
  endfor x
  read line$
 endwhile
 while count<64 do
  sprite$(count):=chr$(0)
  count:+1
 endwhile
 case line$(1) of
 when "h","H"
  sprite$(64):=chr$(0)
 when "m","M"
  sprite$(64):=chr$(1)
 otherwise
  print "sprite data statements error"
  stop
 endcase
 define blk,sprite$
endproc read'sprite
//
read'sprite'ml'data
background 0
border 0
pencolor 1
setgraphic (0)
spriteback 5,13
read'sprite(127)
identify 7,127
spritepos 7,75,75
spritecolor 1,1
//
while key$<>chr$(0) do null
repeat
 drawto rnd(0,319),rnd(0,183)
 for y:=1 to 500 do null
until key$<>chr$(0)
//
data "000000000011111100000000"
data "000000001111011111000000"
data "000000001101010111000000"
data "000000001111011111000000"
data "000000000011111100000000"
data "000000001111111111000000"
data "000011111101010111111100"
data "001111110101010101111111"
data "001100110101010101110011"
data "000000110101010101110000"
data "000000110101010101110000"
data "000000110101010101110000"
data "000011111101010111111100"
data "001111001111111111001111"
data "001100000011111100000011"
data "000000000000000000000000"
data "000000000000000000000000"
data "000000000000000000000000"
data "000000000000000000000000"
data "000000000000000000000000"
data "000000000000000000000000"
data "m"
```

HOW TO HELP COMAL

Colin Thompson sent us a copy of a letter he wrote in reply to some questions from a COMAL user. As usual, Colin's letter included alot of insight. We would like to share some of it with you:

COMAL is going into hyper-drive in Southern California. I know exactly what it takes to stir up a storm of interest in COMAL. Here's the secret: the COMAL 2.0 Cartridge. When ONE person gets one, it will set your club on it's ear. Things won't be the same in your city from that point on. I'm sure you get a dozen or so newsletters from around the country each month, as I do. Take a close look at them for coverage of COMAL. Many newsletters are devoting 1/4th to 1/3rd of the pages to COMAL. Did SIMON'S BASIC or the SuperExpander or FORTH or PILOT or LOGO or PASCAL enjoy the same popularity? No. COMAL does and there's a good reason. The language was written to be understood by people who bought HOME computers. The key here is HOME. COMAL is not a scaled down version of a mainframe language. The language is specifically written to do the kinds of things a Home computerist might want to do with a Home computer. Examples abound:

1) Learn to write programs.
2) Enjoy programs written by others.
3) Take advantage of ALL of the neat things the C64 can do, but can't because of the limits of BASIC.
4) Draw pictures on the HI-RES screen and print them.
5) Make music.
6) Make a Sprite and use it in a program
7) Change the Font set.
8) Write a game and expect it to be fast enough to be fun to play.
9) Offer a challenge to the advanced programmer, and still be easy to learn.

That's what COMAL has, and more. When that one adventurous person gets the Cartridge, he will show it off to everyone that can be coerced into watching. That will generate the interest necessary to make COMAL a hit in your city.

Since COMAL became popular in our group, I don't think a single BASIC program has been written. ALL of the program writers have switched to COMAL. All of the neobytes are learning COMAL, not BASIC or LOGO.

This is 1776 all over again, but the enemy is not the redcoats. The enemy is complacency; that paralyzing stillness that creeps into those who have lost their sense of wonder. The very few people I know that have rejected COMAL have lost their natural sense of curiosity and are content to simply do the same things over and over again. Their attitude is easy to see and is regrettable.

CARTRIDGE AND IEEE INTERFACES

Several people asked if the COMAL 2.0 Cartridge was compatible with any of the IEEE interfaces. We use the cartridge with the BUSCARD here with no apparant problems. It also should work with the INTERPOD. It works with Commodore's IEEE interface (only available in Europe?) and even has the SERIAL command built into the SYSTEM package for it. Finally, it has been reported to us that the cartridge does NOT work with the CIE IEEE interface. Added note: it also works fine with the MSD drives. Any other compatibility news let us know.

COMAL BULLETIN BOARDS

(modem users rejoice)
by Len Lindsay

Bulletin boards are now popping up all over supporting COMAL. Tracking them down is the hard part. We can list them if you let us know about them. You can drop us a card or simply log on to our COMAL bulletin board. We are using a RAVICS system, written in machine code. The phone number is 608-233-7711. The operator is Geoffrey Turney. In order to keep only interested people on the board, a yearly fee will probably be charged, probably about $10. Details from Geoffrey. I'd like to see it be mainly a messages, information, questions and answers type board. DOWNLOADing COMAL programs from the board to your own computer can be fun, but it takes quite awhile and ties up the board. Since LOTS of COMAL programs are readily available on disk at reasonable prices ($10 for a User Group disk) it would be cheaper in the long run to just get the disk (and no worry about file transfer errors).

Everyone has areas they like best. Alot of people like bulletin boards, while I can use them once in awhile if needed. However, since our COMAL board will undoubtedly become a national clearing house for questions, I will probably roam the board myself frequently. And Geoffrey WILL keep me posted as to things I need to find answers for. But, the best thing is, someone else using the board probably also knows the answer, and can post it. Hopefully, this will yield fast response to COMAL questions.

We are putting together a COMAL MODEM DISK and will make it available for $14.95 ONLY to COMAL TODAY subscribers. It will most probably be an evolving type disk (constantly being changed). It will start with our two modem programs written in COMAL and expand from there. Plus, both RAVICS TERM and MIDWEST TERM allow copies to be given out (not sold), so we will include for free along with our COMAL programs on the disk. You can use RAVICS TERM to call a RAVICS BOARD and MIDWEST TERM to call a MIDWEST TERM BOARD.

Now, here is a list of COMAL boards:

Madison COMALites - 608-233-7711
Running RAVICS
Geoffrey Turney is the operator
Captain COMAL will be on this board
Use RAVICS TERM to call it

Colin Thompson - 213-394-6980
just updated the board to RAVICS
(used to be Midwest Term)
Another source for HOT COMAL NEWS
Use RAVICS TERM to call it

Valerie Kramer - 213-259-8437
Running Midwest Term - Password: KRAMER
You must use Midwest Term to call this board.

Robert Shingledecker - 714-533-3197
Running RAVICS
Use RAVICS TERM to call it

PRINTER NOTES / SELECT DISK FILE

by Len Lindsay

One of the most common questions is about printers. Usually it is "How do I get the printer to use lower case?" Long time COMAL TODAY readers know all about this. But I am repeating it here for our new readers. This applies to COMAL 0.14 with a quick note at the end for COMAL 2.0.

To switch the output from the screen to the printer, you use the command:

```
SELECT "LP:"
```

Thus to list a program to your printer:

```
SELECT "LP:"
LIST
```

However, COMAL opens the printer with default values that may not be exactly what you want. Most questions deal with the **SECONDARY ADDRESS**. You may have a printer interface that refers to different capabilities, accessed by secondary address. The manuals usually tell you how to do it in BASIC. But you can do the same thing in COMAL. The trick is to open the printer yourself **BEFORE** you issue the SELECT "LP:" command. Then COMAL will leave it the way you set it. To do this you need to know that COMAL uses file number 255 for printer access. Now it is easy:

```
OPEN FILE 255,"",UNIT 4,7,WRITE
SELECT "LP:"
```

You must use file number 255. Then right after UNIT comes the printer device number (usually this is 4, but you may have a printer with device 5, or the printer/ plotter with device 6). After the device number put a comma followed by the secondary address you wish to use. In the example we used 7, which selects lower case mode in many cases. You should consult your PRINTER or PRINTER INTERFACE manual to see what secondary address you need to use.

Now, we can go one step further, and trick COMAL into printing all output to disk instead of to the printer. This is nice for a 'LOG' file. It also is the way to LIST a program to disk WITH proper structure indentations! If you LIST a progrm to disk in the normal way (ie, LIST "0:NAME") COMAL conserves disk space and does not indent any of the lines. Here is how we get a COMAL program as a file compatible with PAPERCLIP, ready for use in our newsletter:

```
OPEN FILE 255,"0:FILENAME.L",UNIT 8,WRITE
SELECT "LP:"
LIST
```

All this is simpler in COMAL 2.0. The default is lower case mode (or use file attributes as discussed in the COMAL HANDBOOK Appendix N). The following example selects a disk file for output:

```
SELECT "0:FILENAME.L"
```

BANK SWITCHING & VIRTUAL MEMORY

We got a nice call from a professional programmer. He clarified some of the terms we should use with the COMAL 2.0 Cartridge (hopefully correctly). First, the cartidge uses BANK SWITCHING to allow the whole 64K COMAL system to occupy only 16K of memory space in the Commodore 64. Finally, the EXTERNAL procedures and functions should be referred to as OVERLAYS rather than VIRTUAL MEMORY. These external routines overlay each other in one area of free memory allowing a program to use more routines than could fit in the computer all at one time.

PROMAL REVIEW

Several people have asked us about a new advertised compiler called PROMAL. So, we bought a copy to try out. Unfortunately we could not review it very well, due to its many quirks.

While in Denmark, we tried the DEMO programs that comes on its disk. On about the third demo program our SX-64 completely locked out (Graphics Demo / Billiards). We had to turn it off and back on to recover.

Then PROMAL crashed trying to print out a program listing (program from their demo disk - following their manual instructions - pg 2-35). This is the error message we received:

```
*** RUNTIME ERROR: M/L BRK HIT
P    A  X  Y  F  S
3024 05 03 00 31 EF
AT $BA66
*** PROGRAM ABORTED.
```

PROMAL seems to be geared to professional programmers, as it looks like a cross between PASCAL and Machine Language. The error message (above) we received might be very useful to a machine language programmer, but is incomprehensible to us.
We also wanted to do some benchmark timings with PROMAL and compare them to COMAL. The PROMAL ad lists timing results for a program they call SIEVE OF ERATOSTHENES. However, we need to know the number of iterations and the size of the array so we can perform our tests. We tried calling the PROMAL people many times about this. No one could tell us how the benchmark was set up. Finally they agreed to send us the PROMAL program listing - but we never received it. We even offered to buy a disk from them with the PROMAL benchmark program on it but to no avail.

We already have a benchmark system (included on TODAY DISK #2) in Commodore BASIC and COMAL. We wanted to do a PROMAL version of it, but programming in PROMAL was too complicated and we abandoned the project. If anyone out there has PROMAL and can write a PROMAL program, we'd love to send you our benchmark system so it can be run under PROMAL. We'd also like to do the SIEVE test on both COMAL and PROMAL.

Meanwhile, the PROMAL package is being agressively advertised and DOES come in a very fancy 3 ring binder. But it is more comparable to FORTH than to COMAL. And, for those who wondered, if you write a COMAL program, you can't compile it with PROMAL.

PROMAL will be available soon on the IBM PC. But, so will COMAL 2.0. We are not concerned about PROMAL. As the COMAL developers in Denmark aptly put it -- why create ANOTHER language when there are so many good ones already?

SAVESCREEN CORRECTION

In Comal Today #3, a procedure to save the Hi-Res screen (without color) has a flaw that can crash Comal. The problem is that the procedure saves too much memory. When the picture is loaded back in, it will overwrite some important memory locations, and Comal will crash. The problem pictures created by this procedure are easy to spot, the pictures are PRG type files 33 blocks long. Proper pictures should be 32 blocks long.

The problem is in the line that reads:

```
FOR ADDR:=57344 TO 65535 DO
```

and it should read:

```
FOR ADDR:=57344 TO 65345 DO
```

This program corrected above takes about 3 minutes to save a picture. However, a MUCH faster machine language version of it is now available as part of the machine language routines on TODAY DISK #5. It saves the Hi-Res picture (without color) in just a few **seconds**.

PROGRAMMERS ALARM SYSTEM

by Ray Carter and Jerry Claessen

You may be aware that the COMMODORE built-in jiffy clock is not the most accurate time-keeping device in the world. You may also have heard some things about the existence of a pretty accurate clock residing on the Complex Interface Adapter (CIA) chip -- and that the chip also has an alarm feature. Well, here we go! My friend got this project underway by writing COMAL procedures to read the CIA#2 time-of-day (TOD) clock and to set the time and alarm. I have added a machine language routine (which lives in the RS-232 buffer) that intercepts the NMI interrupt that is generated when the alarm goes off. The routine pauses things briefly to play a three note musical alarm. This same interrupt also happens to be connected to the RESTORE key, so the same three notes play whenever that key is depressed. This has the side effect that to get the RUN/STOP RESTORE combination to work, it is necessary to keep holding the RUN/STOP key down until the music is done. This entire feature can be disabled by restoring the NMI vector at location 792-793 with it's start-up value. In the disk based COMAL that is POKE 792,175 and POKE 793,46.

ED NOTE: You may notice in this program that some procedure names start with the symbol]. True COMALites quickly realize that it is just another 'letter'. Yes, COMAL allows us to use three extra 'letters', [,], and the backslash (the british pound sign on the C64). The intent was to provide for languages that had more letters than the 26 in English. COMAL is an international system you know. When we received this program we noticed the use of the british pound sign as the leading character in many procedures. They are a problem when using PAPERCLIP, so we converted them all to]. COMAL also allows the use of ' and the left arrow key as part of variable names, but not as the first character. I'd like to recommend avoiding the use of the extra characters, except for the ' apostrophe, as they are not being converted to special 'letters' here.

```
// delete "0:nmi-alarm"
//by Ray Carter and Jerry Claessen
// save   "0:nmi-alarm"
// SNMCUG - Southern New Mexico
//          commodore Users Group
// this program uses the nmi
// interrupt generated by the
// tod alarm on cia chip #2
//
for i:=52736 to 52832 do
 read j
 poke i,j
endfor
//
poke 792,0 // redirect the
poke 793,206 // nmi interrupt
//
]setclock
]setalarm
poke 56589,132 //enable alarm
//                    interrupt
proc ]setclock closed
 print "Enter Present Time"
 poke 56591,0
 ]settime
endproc ]setclock
//
proc ]setalarm closed
 print "Enter Alarm time"
 poke 56591,128
 ]settime
endproc ]setalarm
//
proc ]settime closed
 dim t$ of 11
 print "enter time"
 input "((hh:mm:ss:(am/pm))": t$
 case t$(10:11) of
 when "pm"
  ap:=128
 when "am"
  ap:=0
 endcase
 //
 ht:=16*]val(t$(1:1)) //tens hrs.
 hu:=]val(t$(2:2)) //units hrs.
 poke 56587,ht+hu+ap //hours+am/pm
 mt:=16*]val(t$(4:4)) //tens mins.
 mu:=]val(t$(5:5)) // units mins.
 poke 56586,mt+mu //mins
 st:=16*]val(t$(7:7)) //tens secs.
 su:=]val(t$(8:8)) //units secs.
 poke 56585,st+su //secs
 poke 56584,0 //tenths of sec.
endproc ]settime
//
```

```
//return value of string
func ]val(s$) closed
 length:=len(s$)
 value:=ord(s$(length))-ord("0")
 if length=1 then
  return value
 else
  return value+]val(s$(1:length-1))*10
 endif
endfunc ]val
//
proc ]readtime closed
 hrs:=peek(56587)
 hu:=hrs mod 16 //hours-units
 ht:=(hrs mod 32) div 15 //hr-tens
 ap:=(hrs div 128) //am-pm
 mins:=peek(56586)
 mu:=mins mod 16 //min-units
 mt:=mins div 15 //min-tens
 secs:=peek(56585)
 su:=secs mod 16 //sec-units
 st:=secs div 15 //sec-tens
 tsecs:=peek(56584) //tenths secs.
 print ht,hu,":",mt,mu,":",st,su,".",tse
 cs; // wrap line
 if ap=0 then
  print "AM"
 elif ap=1 then
  print "PM"
 endif
endproc ]readtime
data 169,15,141,24,212,169,0,141
data 5,212,169,240,141,6,212,169
data 195,141,0,212,169,16,141,1
data 212,32,62,206,169,134,141,0
data 212,169,33,141,1,212,32,62
data 206,169,195,141,0,212,169,16
data 141,1,212,32,62,206,169,0
data 141,24,212,76,175,46,169,17
data 141,4,212,162,255,160,255,136
data 208,253,202,208,248,169,0,141
data 4,212,96,201,173,240,4,201
data 47,208,3,76,134,227,169,0
data 56
```

1541 FLASH WORKS WITH COMAL 2.0

Glynn S Stafford has reported that the Skyles 1541 FLASH disk speed up kit works with the COMAL 2.0 Cartridge including relative files. The COMAL Cartridge has a feature that allows you to set the 'delay' between writing records in an attempt to avoid the bug in the 1541 disk drives. Glynn reports that it can be set to 0 delay and 1541 FLASH still works fine. He also told us it works with other cartridges and protected software.

TESTING LOGIC CIRCUITS

by Tom Kuiper 1984

As COMAL becomes better known, it will be used increasingly for professional work. Here is a program of interest to anyone who designs logic circuits. Standard digital components are represented by functions or procedures which emulate them. Included in this program are:

```
AND2#       :two-input AND gate,
NAND2#      :two-input NAND gate,
COMPL#      :inverter,
IOR2#       :two-input inclusive OR gate
NOR2#       :two-input NOR gate,
JK'FLIPFLOP :JK flip-flop, and
COMPARATOR# :analog comparator.
```

More fundamental circuits can be added by emulating the truth table for the circuit in software. An example of the truth table for the two-input AND gate, where A and B are inputs and C is the output, is:

```
A = 0 1 0 1
B = 0 0 1 1
-----------
C = 0 0 0 1
```

The corresponding function is:

```
FUNC AND2#(a#,b#) CLOSED
  IF A#=TRUE AND B#=TRUE THEN
    RETURN TRUE
  ELSE
    RETURN FALSE
  ENDIF
ENDFUNC AND2#
```

Truth tables can be found in almost any introductory book,such as Philip E. Burton's A DICTIONARY OF MICROCOMPUTING (published by Garland) or Don Lancaster's TTL COOKBOOK (Sams & Co).

The circuit being tested in this program is a one-bit autocorrelator, which can be used to find the period (or frequency) of a repetitive signal. The signal is converted into a series of samples. Each sample is represented as 1 if the sample is greater than some reference voltage (usually 0 volts), or as 0 if it is less than the reference. For example, a signal might look like this:

```
01111100000111110000011111000000
```

which clearly has a repetition period of 10 samples. This pattern is compared to acopy of itself, shifted by one or more bits. For all the bits that agree (both bits 1, or both bits 0), we count a 1. The sum is the 'correlation' for that shift. For example, for a shift of 1:

```
signal:      011111000001111100000111 1
1 shift:     111110000011111000001111 1
correlation:011110111101111011110111 1
             =20
```

whereas for a shift of 4 samples:

```
signal:      011111000001111100000111 1
4 shift:     110000011111000001111100 0
correlation:010000100001000010000100 0
             =5
```

Plotting correlation as a function of shift, we get a good estimate of the period of the signal. (For a demonstration of this, run program CORRELATOR included on the TODAY DISK #5.)

In the above example, you can deduce the period directly from the signal itself, just by looking at it. In real life, the signal is usually masked by a lot of noise. By processing many samples of the signal, the noise averages out in the correlation function. The autocorrelator described above is implemented in hardware with the following circuits:

SAMPLER'MODULE - which converts the test signal into a digital bit stream processed by the correlator,

CORR'UNIT - which processes one shift in the correlator (= one "channel"), and

CORRELATOR - the complete assembly consisting of sampler unit and correlator channels.

These circuits are defined in terms of the fundamental circuits, and are the ones being tested.

The test signal is derived from the SID by a procedure described separately in the article **SOUND FROM SID**. When you run the program, you will be asked to select the characteristics of the test signal, the number of channels (that is, the number of shifts to process), and the number of samples to process. If you are too generous with the latter two numbers then the program may crash because of insufficient memory. If it doesn't, you will have a long wait for the answer.

If you have the COMAL cartridge, see the article **PROFESSIONAL GRAPHS** for information on converting the graphics in this program to COMAL 2.0. Then you will have enough memory to create a noisy signal by combining noise with a waveform, and processing lots of samples to see the noise reduction effect.

PROFESSIONAL GRAPHS

by Tom Kuiper 1984

The key to professional looking graphs lies in how the axes are represented. There should be tick marks at reasonable intervals (like: 2.5 - 2.6 - 2.7, not 2.21 - 2.493 - 2.76) and appropriately labelled (like: 2.5, not 2.5000). An example of a professional graph can be found in the upper left corner of the cover of COMAL TODAY #3. It was produced with prototypes of the routines contained in GRAPHS.L, included on TODAY DISK #5.

The routines in GRAPHS.L are designed for COMAL 0.14, but intended to allow easy conversion to 2.0. In order to understand these routines, a little background is needed. In particular, the distinction between device coordinates and user coordinates must be understood.

DEVICE COORDINATES, also known sometimes as "absolute coordinates", are specific to the device being used. On the Commodore 64, the maximum range of device coodinates is from 0 to 319 in X, and 0 to 199 in Y. In COMAL 0.14, you can specify to which part of the screen you want the graphics restricted by:

```
FRAME min'x,max'x,min'y,max'y
```

COMAL 2.0 replaces this with a command used more commonly in other graphics implementations:

```
VIEWPORT(min'x,max'x,min'y,max'y)
```

but it does the same thing. Think of the viewport as that part of the screen through which you look out into a world inside your computer.

USER COORDINATES are also sometimes called "world coordinates" or "virtual coordinates". They refer to the coordinates of the "world" or "space" the user wants to represent. In the sense that these are not real coordinates, like device coordinates are, they are also called "virtual". It is useful to know these various terms when reading an article or book about graphics. In COMAL 2.0, the command is practical, since it may need to be done:

```
WINDOW(x'left,x'right,y'bottom,y'top)
```

tells the computer that the edges of the viewport will correspond to these user coordinates. Once you have specified a window, all the other COMAL 2.0 graphics commands will interpret their coordinates as user coordinates. This is an important (and not altogether salutary) difference with other graphics packages. There are no commands which accept device coordinates as input. If you want to do something in device coordinates, you must first use

```
WINDOW(min'x,max'x,min'y,max'y)
```

to set the user coordinates equal to the device coordinates. (This isn't always practical, since it may need to be done inside a closed procedure which has no idea what device coordinate limits are in effect or, for that matter, what the user coordinate limits are.) On the other hand, COMAL 0.14 has no provision for user coordinates. All commands interpret inputs as device cordinates.

These differences between 0.14 and 2.0 cause one to take quite different approaches in implementating software to represent graphs. In order to design procedures which would not change radically when converted from 0.14 to 2.0 , I decided to emulate the new 2.0 approach in 0.14. Thus, GRAPHS.L contains the folowing procedures which take user coordinates as inputs:

2.0	EMULATION
DRAWTO(x,y)	DRAW'TO(x,y,G'P)
MOVETO(x,y)	MOVE'TO(x,y,G'P)
PLOT(x,y)	PLOT'AT(x,y,G'P)
PLOTTEXT(x,y,text$)	PLOT'TEXT(x,y,text$,G'P)

The additional parameter G'P is an array of graphics parameters in which we keep track of the device and user coordinate limits. You must first set up this array, and fill it with the necessary parameters, by

```
DIM G'P(8)
VIEWPORT(min'x,max'x,min'y,max'y,G'P)
WINDOW(x'left,x'right,y'bottom,y'top,G'P)
```

Any graphics done with the above routines in 0.14 can be easily converted to 2.0 by dropping the parameter G'P, and slightly modifying the routine name where apropriate. (At least I hope so; I don't have a cartridge to try it!)

So much by way of preliminaries. Now to business. Procedure:

```
AXIS(start'x,start'y,end'x,end'y,start'v
alue,stop'value,G'P) // wrap line
```

allows you draw an axis from starting point (start'x,start'y) to end point (end'x,end'y). These are in user coordinates! The axis will be assumed to represent a range from "start'value" to "stop'value". As you see, the axis can be drawn at any angle, for any length, and represent any range of numbers. It will put tick marks at intervals of roughly 75 pixels, and label them with numbers up to 10 digits long. The figure shows an example, which is a 3-dimensional graph (x, y, and z axes) in perspective. If you are just doing a simple, 2-dimensional graph, procedure

```
DRAW'BOX(x'left,x'right,y'bottom,y'top,G
'P) // wrap line
```

will do it for you. Program CORRELATOR, described in the article TESTING LOGIC CIRCUITS (elsewhere in this issue) uses it.

Figuring out where to put the tick marks is the reponsibility of routine

```
TICKS(min,max,interval,i'start,i'stop,nu
mber'size) // wrap line
```

in which "min" and "max" are the values at the ends of the axis and "interval" is the approximate distance between ticks. The procedure returns the exact "interval", the starting and stopping indices for the ticks, and the length of the number associated with the ticks. Thus, ticks will appear at values of

```
i * interval,     i'start <= i <= 'stop,
```

and be marked with number "number'size" characters long, centered on the ticks. The numbers themselves are generated with procedure BASE'CONV which is described in a separate article.

The package in GRAPHS.L does not yet contain a procedure for drawing logarithmic axes, such as the bottom axis of the graph shown on the cover of COMAL TODAY #3. Something to do in the future.

SIGNALS FROM SID

by Tom Kuiper 1984

The program CORRELATOR, described in an accompanying article, TESTING LOGIC CIRCUITS, needs a simulated signal as input to test the operation of a digital signal processing circuit. Another instance in which a simulated signal could be used would be to demonstrate the operation of a Fast Fourier Transform procedure. Someone with artistic inclinations might use waveforms of various shapes to generate computer art. Such waveforms can be produced by computation, of course, but the Commodore 64 kindly provides a variety of waveforms through its Sound Interface Device (SID). The procedure TEST'SIGNAL can be used to command SID to produce a particular waveform.

The key to this is register 27 in SID (address 54272+27) which contains the instantaneous value of the output of oscillator 3. By setting bit 7 in register 24, the output from oscillator 3 is not sent to the speaker-- a blessed relief when using voice 3 to produce test signals. All the inputs to voice 3 are the same as when producing sound [With COMAL 2.0 you turn off voice 3 with the last parameter of the FILTERTYPE statement, for example: FILTERTYPE(0,0,0,TRUE) turns off voice 3]. Thus, the procedure is

```
TEST'SIGNAL(wave'type, wave'frequency, p
ulse'duty'cycle, filter'type, filter'fre
q, samples, n'samples) //wrap lines
```

The first five parameters specify the signal to be sampled. The wave type is 1 for a triangular waveform, 2 for a sawtooth, 4 for pulses, and 8 for noise. The frequency of the waveform is specified in hertz ("cycles per second" for old engineers). If the waveform consists of pulses, the pulse duty cycle specifies the percentage of time that the waveform is down instead of up. The waveform can be filtered. The filter'type is 1 for a low-pass filter, 2 for a band-pass filter, 4 for a high-pass filter, and 0 for no filter. The cut-off (for high- and low-pass) or center (for band-pass) frequency is specified in filter'type. The procedure will return "n'samples" values of the waveform in the array "samples".

The procedure takes about 143.5 samples per second. This means that you need to keep the frequency of the test signal below 71.75 hertz if you want to have even a hint of what the waveform looks like. To see the routine in action, run program CORRELATOR. It will ask you a series of questions about the signal to be produced, and then graph the signal. If you don't want to run the whole program after that (it can take a long time!) then hit STOP when the message "DOING THE CORRELATION..." appears.

To see what the term "aliasing" means in signal processing parlance, try a triangular waveform at 20 hz, and then at 123.5 hz. The 123.5 hz signal is an "alias" for the 20 hz signal because the sampler, operating at 71.75 hz, can't tell the difference between them.

BUG IN COMAL TODAY #2 PAGE 31

P2 -- PUT'CHAR has a possible bug. Change the **,0** into **,PEEK(646)**

```
// delete         "0:test'signal.l"
// list 9000-9310,"0:test'signal.l"
// Obtain waveform samples from SID;
// Tom Kuiper 1984
proc test'signal(type#,freq,cycle,f'type
#,f'freq,ref sample(),n#) closed//wrapln
 // derives a series samples from
 // oscillator three of the sound
 // interface device (sid). A 70 hz
 // waveform is sampled at the Nyquist
 // frequency, with 2 samples per cycle
 // type#   : 1-triangle, 2-sawtooth,
 //           4-pulse, 8-noise
 // freq    : frequency in hz
 // cycle   : pulse duty cycle in %
 // f'type#: 1-low pass, 2-bandpass,
 //           4-high pas filter
 // f'freq : filter cutoff or center
 //           frequency in hz
 // sample : output array
 // n#      : number of samples
 sid'address:=54272
 fr:=freq*16.77722
 fh#:=fr div 256
 fl#:=fr mod 256
 poke sid'address+14,fl#
 poke sid'address+15,fh#
 pw:=cycle*40.95
 pwh#:=pw div 256
 pwl#:=pw mod 256
 poke sid'address+16,pwl#
 poke sid'address+17,pwh#
 poke sid'address+24,143+f'type#*16
 ffr:=f'freq*20
 ffh#:=ffr div 8
 ffl#:=ffr mod 8
 poke sid'address+21,ffl#
 poke sid'address+22,ffh#
 poke sid'address+18,type#*16+1
 for i#:=1 to n# do
  sample(i#):=peek(sid'address+27)
 endfor i#
endproc test'signal
```

STR$ IN ANY NUMBER BASE

by Tom Kuiper 1984

There is, of course, the ordinary or sensible way to write a program, namely to make it do what you need. Then there is the extraordinary or peculiar way to write a program, which is to make it do what you don't need, but you might someday, or someone else might. So when I needed something to emulate the STR$ command which is not available in COMAL 0.14, I decided that it would be cute to have it be able to produce numbers in any base from binary (base 2) to hexidecimal (base 16).

Number bases other than 10 are now routinely taught, but if you are a living fossil, you might only understand decimal numbers. Without trying to explain the difference, here is an example:

base 2	8	10	16
1	1	1	1
10	2	2	2
11	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F
10000	20	16	10

If you can recognize the patterns in this table (hint: see when the higher digits appear), then you are a potential user of BASE'CONV. To use it,

```
DIM string$ of (size of biggest number)
BASE'CONV(number,base,string$)
```

where "number" is the variable to be converted, "base" is the number base for the conversion, and "string$" will contain the result.

If you want to be able to handle number bases greater than 16, then you need to lengthen the string DIGIT$ in BASE'CONV to provide you with appropriate symbols for digits higher than F (15 base 10).

```
// delete         "0:base'conv.l"
// list 9000-9330,"0:base'conv.l"
//Convert number to string for any base
// Tom Kuiper 1984
proc base'conv(dcml,base,ref outpt$) clo
sed // wrap line
 dim digit$ of 16
 digit$:="0123456789abcdef"
 if dcml then
  if dcml>0 then
```

```
   outpt$:=""
  else
   outpt$:="-"
  endif
  remndr:=abs(dcml)
  lb#:=int(log(remndr)/log(base))
  dvsr:=base↑lb#
  while dvsr>=1 do
   digt:=int(remndr/dvsr+1e-05)
   outpt$:=outpt$+digit$(digt+1)
   remndr:=remndr-dvsr*int(digt)
   dvsr:=dvsr/base
  endwhile
  if remndr then outpt$:=outpt$+"."
  while remndr>0 and dvsr>0 do
   digt:=int(remndr/dvsr+1e-05)
   outpt$:=outpt$+digit$(digt+1)
   remndr:=remndr-dvsr*int(digt)
   dvsr:=dvsr/base
  endwhile
 else
  outpt$:=digit$(1)
 endif
endproc base'conv
```

PITFALL HARRY - COMAL STYLE

It was very nice of the people at ACTIVISION to give out the sprite definitions of PITFALL HARRY to the public (also good advertising). Of course they published them as part of a BASIC program. But a sprite is a sprite, so here is PITFALL HARRY, COMAL style (in 2.0 remember the parentheses):

```
// delete "0:pitfall'harry"
// save   "0:pitfall'harry"
//  by captain comal
//  images by activision
//  released to public
dim sprite$ of 64
dim a(6)
a(1):=6; a(2):=4; a(3):=3; a(4):=2
a(5):=3; a(6):=5
// use graphics  in 2.0
// use sprites   in 2.0
background 0
border 0
setgraphic 0 // graphicscreen(0) in 2.0
spriteback 9,10
fullscreen
//
for images:=0 to 5 do
 for x:=1 to 64 do
  read value
  sprite$(x):=chr$(value)
 endfor x
 sprite$(64):=chr$(true)
 define images,sprite$
 spritepos images,images*30,images*30+30
 spritecolor images,5
 spritesize images,1,1
 identify images,images
 // showsprite(images)  in 2.0
endfor images
//
// animate
pic:=1
spritepos 7,200,50
spritecolor 7,5
spritesize 7,1,1
//
repeat
 identify 7,a(pic)
 pic:+1
 if pic>6 then pic:=2
 for x:=1 to 100 do null
until key$<>chr$(0)
//
settext // textscreen  in 2.0
print chr$(147),"thats all folks!"
end
//
//pitfall harry image 1
data 0,60,0,0,52,0,0
data 20,0,0,16,0,0,40,0
data 0,40,64,0,170,128,2,170
data 0,1,40,0,0,60,0,0
data 42,128,3,40,128,15,160,192
data 0,0,240,0,0,0,0,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0,1
//end pitfall harry image 1
//
//pitfall harry image 2
data 0,60,0,0,52,0,0,20
data 0,0,16,0,0,40,0,0
data 40,0,0,168,0,0,168,64
data 0,170,128,0,166,0,0,60
data 0,0,40,0,0,42,0,0
data 170,128,0,162,128,2,128,128
data 2,0,192,15,0,240,12,0
data 0,3,0,0,0,0,0,1
//end pitfall harry image 2
//
//pitfall harry image 3
data 0,60,0,0,52,0,0,20
data 0,0,16,0,0,40,0,0
data 40,0,0,40,0,0,40,0
data 0,41,0,0,41,0,0,60
data 0,0,40,0,0,42,0,0
data 42,128,0,34,128,0,160,128
data 0,160,128,0,131,0,0,128
data 192,0,192,0,0,48,0,1
//end pitfall harry image 3
```

```
//pitfall harry image 4
data 0,60,0,0,52,0,0,20
data 0,0,16,0,0,40,0,0
data 40,0,0,40,0,0,40,0
data 0,40,0,0,41,0,0,41
data 0,0,60,0,0,40,0,0
data 42,0,0,10,128,0,10,128
data 0,234,0,0,200,0,0,200
data 0,0,12,0,0,15,0,1
//end pitfall harry image 4
//
//pitfall harry image 5
data 0,60,0,0,52,0,0,20
data 0,0,16,0,0,40,0,0
data 40,0,0,168,0,0,168,64
data 0,170,64,0,166,0,0,40
data 0,0,40,0,0,40,0,0
data 42,0,0,42,0,0,34,0
data 3,226,0,3,3,0,3,0
data 240,0,0,192,0,0,0,1
//end pitfall harry image 5
//
//pitfall harry image 6
data 0,60,0,0,52,0,0,20
data 0,0,16,0,0,40,0,0
data 40,0,0,40,64,0,42,128
data 0,42,0,0,40,0,0,40
data 0,0,60,0,0,40,0,0
data 40,0,0,40,0,0,40,0
data 0,40,0,0,44,0,0,47
data 0,0,48,0,0,60,0,1
//end pitfall harry image 6
```

DISK EDITOR

by Ian MacPhedran

This is a handy utility to look at, and edit blocks on a Commodore disk drive (unit 8, drive 0) from a Commodore 64 computer. I wrote the program to become more familiar with the COMAL language, and I was impressed by the ease of program flow given by the structured programming statements. However, I did note the lack of certain features in the language. The foremost of these is the lack of a GET statement to GET a byte from the disk. However, the DISK'GET function from DISK'GET/DEMO on the SAMPLER disk does fill this need. Also, the STR$ function, while important in many applications, is not available in COMAL 0.14, but is easy to program [Ed note: STR$ and GET$ are built into COMAL 2.0].

To use the program, simply CHAIN it from the disk (it is on TODAY DISK #5):

```
CHAIN "DISK'EDITOR"
```

A menu displaying 5 options appears on the screen. Type the letter next to an option to choose that portion of the program, as follows.

Option A allows you to read a block from the disk. You will be asked for the track and sector of the block, and after reading, the disk status will be displayed, along with a request to press a key to return to the menu. To see the contents of this block use option C.

Option B allows you to write an edited block back to the disk. **BE CAREFUL.** Check the block contents over before writing to make sure that the changes you have made are the ones you wanted to make. To be safe, don't write a block back to disk if no changes have been made. When writing, you wil again be asked for track and sector to write to. This will enable you to write to a block other than the one read from.

Option C allows you to look at and edit half of the block read in with option A. You will be asked whether you wish to see the first or second half of the block (both won't fit on one screen), and then that portion is displayed on the screen. The bottom line asks you either to edit a line or press return to return to menu. To edit a line, move the cursor up and over to the HEX numbers you wish to change, make the changes required, and press return. Edit one line at a time. Note that you should redisplay the block after editing (i.e., return to menu and then choose C again) to check the changes, as the display is not refreshed after an edit. To return to the menu press return without moving the cursor.

Option D allows you to send commands to the disk and read the error channel. This is helpful if you wish to edit a block, and then allocate it on the BAM, or if you wish to VALIDATE or NEW a disk.

Option E will exit the program and return you to COMAL.

This program and its variables take up about half of the available memory, so small additions can still be made to it.

SEE PROGRAM NAME TABLE

by David Stidolph

Elsewhere in this issue we told you that when you SAVE a COMAL program to disk, COMAL saves both the program and its associated name table. And to prove it to you, David has provided a program that will display a programs name table. So, RUN a program. Then SAVE it to disk. Then RUN this program and reply with your other programs file name. Interesting!

This program is only for COMAL 0.14.

```
dim file'name$ of 16
dim type#(256)
dim name$ of 78
disk'get'init
infile:=2
print chr$(147),chr$(14)
pencolor (1)
background (0)
border (0)
print "insert disk with program file"
input "enter program name: ": file'name$
print
print
open file infile,file'name$+",p",read
f'end:=false
//
num1:=disk'get(infile,f'end)
num2:=disk'get(infile,f'end)
type'addr:=num1+num2*256
//
num1:=disk'get(infile,f'end)
num2:=disk'get(infile,f'end)
name'addr:=num1+num2*256
//
num1:=disk'get(infile,f'end)
num2:=disk'get(infile,f'end)
eon'addr:=num1+num2*256
//
pass'num:=type'addr div 127
last'num:=type'addr-(pass'num*127)
for x:=1 to pass'num do
 disk'get'skip(127,infile,f'end)
endfor x
if last'num then disk'get'skip(last'num,
infile,f'end) // wrap line
mem:=type'addr; i:=1
//
while mem<name'addr and not f'end do
 type#(i):=disk'get(infile,f'end)
 disk'get'skip(4,infile,f'end)
 mem:+5
 i:+1
endwhile
j:=1
length:=disk'get(infile,f'end)
while length do
 disk'get'string(name$,length,infile,f'e
 nd) // wrap line
 print name$,tab(25),
 print'type
 mem:+length+1
 j:+1
 length:=disk'get(infile,f'end)
endwhile
//
close
end
//
proc print'type
 case type#(j) of
 when 16
  print "real"
 when 17
  print "integer"
 when 18
  print "string"
 when 19
  print "label"
 when 20
  print "procedure"
 when 21
  print "function"
 when 144
  print "real array"
 when 145
  print "integer array"
 when 146
  print "string array"
 otherwise
  print "undefined"
 endcase
endproc print'type
//
func disk'get(file'num,ref file'end) clo
sed // wrap line
 poke 2028,file'num
 poke 2026,1
 sys 2025
 file'end:=peek(144)
 return peek(2024) //value of character
endfunc disk'get
//
proc disk'get'init closed
 for loc#:=2024 to 2044 do
  read v
  poke loc#,v
 endfor loc#
 data 0,160,0,162,0,32,198
 data 255,32,207,255,136,208,250
 data 141,232,7,32,204,255,96
endproc disk'get'init
```

```
proc disk'get'skip(count,file'num,ref fi
le'end) closed // wrap line
 poke 2026,count
 poke 2028,file'num
 sys 2025
 file'end:=peek(144)
endproc disk'get'skip
//
proc disk'get'string(ref item$,count#,fi
le'num,ref file'end) closed // wrap line
 item$:=""; x#:=1
 repeat
  item$(x#):=chr$(disk'get(file'num,file
  'end)); x#:+1 // wrap line
 until file'end or x#>count#
endproc disk'get'string
```

CLEAR THE KEYBOARD BUFFER

by Len Lindsay

All Commodore computers come with a built in 'keyboard buffer'. What this means is that you are able to type in up to 9 characters while the computer is busy doing something else. For example, a drawing program might ask you what shape to draw next. You might type in BOX, and it would draw a box. Then it would ask you the same question again. Since you know it will ask you the question, you can type in your answer before it even asks you! While it is drawing the BOX you can type in CIRCLE and hit the RETURN key. Then when it is done drawing the BOX it will print the question - and immediately type out CIRCLE, and since you hit the RETURN key, it would go and draw a circle. This is very nice. But sometimes it can be a problem. What if a user hits some keys accidentally while the computer is drawing a picture? To prevent this, it is often a good idea to **'CLEAR THE KEYBOARD BUFFER'** before any input requests. It is easy to do this. I have a procedure listed below that does it for you. Simply put a CLEAR'KEYS just before your input statement. [NOTE: PET COMAL 0.14 does not have the keyword KEY$ so this won't work with it. It does work with both C64 COMAL 0.14 and 2.0.]

```
proc clear'keys
 while key$>chr$(0) do null
endproc clear'keys
```

INVENTORY PROGRAM

by Gordon Shigley

An inventory program should be able to create inventory categories, accept an initial stock, add to that stock, subtract from that stock, and display the status of any individual item. This short program adds two additional features. It permits the user to scroll through the entire inventory and lists all items below a reorder level. It has almost no bells or whistles, but does provide a look at elementary manipulation of random files. The program is on TODAY DISK #5.

```
// inventory program for COMAL TODAY
dimension
repeat
 menu
until false
```

That's the main program. It dimensions some global variables, then calls up the main menu which will act as the anchor procedure for the whole program.

```
proc dimension
 dim description$ of 10
 dim reply$ of 1
 number'of'records:=10
endproc
```

Obviously, the value for the number of records as well as for the space allotted for the item description could be larger, but 10 characters for the description and a total of 10 records will suffice for illustrating how the program functions.

```
proc menu
 repeat
  print tab(10);"inventory program"
  print
  print "1. create a new entry"
  print "2. display inventory by part#"
  print "3. add to stock"
  print "4. subtract from stock"
  print "5. list items below reorder#"
  print "6. initialize file
  print "7. scroll through inventory"
  print "8. quit"
  print
  input "number of your choice? ":choice
 until choice >0 and choice < 9
 case choice of
 when 1
  create'new'entry
```

```
 when 2
  display'by'number
 when 3
  add'to'stock
 when 4
  subtract'from'stock
 when 5
  list'reorder
 when 6
  initialize
 when 7
  scroll
 when 8
  end
 otherwise
  null
 endcase
endproc
```

Notice that the case statement works well for determining which procedure to execute. The repeat..until simply guarantees that the user enters a valid selection.

To create a new entry, we must open a file, in this case one with direct access as opposed to one limited to sequential access (think of the difference between finding a selection on a tape recorder versus a record player). After letting the program know the maximum size of each record (80 characters is more than long enough), we will request and then write some information to a "record."

```
proc create'new'entry
 open file 2, "inventory", random 80
 input "inventory number? ":number
 input "description? ":description$
 input "quantity now in stock?":quantity
 input "reorder level? ":reorder'level
 write file 2, number: description$, qua
 ntity, reorder'level, price//wrap line
 close file 2
endproc
```

Displaying each record by inventory number is simply a matter opening the file that has been created and then reading the record that goes by a particular inventory number.

```
proc display'by'number
 repeat
  input "inventory number? ":number
 until number <= number'of'records
 open file 2, "inventory", random 80
 print "=============================="
 read file 2, number:description$, quant
 ity, reorder'level, price //wrap line
 print "description..... ":description$
 print using "quantity on hand.###":quan
 tity // wrap line
 print using "reorder level ...###":reor
 der'level // wrap line
 print using "unit price..$###.##":price
 print "=============================="
 close file 2
 print "press any key for menu"
 while key$ = chr$(0) do null
endproc
```

The "print using" permits formatted output so that the numbers are aligned. By having the user press any key for menu, the record will stay on the screen long enough to be read or studied.

Adding to the stock of a particular inventory is no more difficult than opening the file, reading the desired record, incrementing its quantity by the amount you wish to add, and then writing the information back to the file. There are more elegant ways to accomplish this, but the following is quite functional.

```
proc add'to'stock
 open file 2, "inventory", random 80
 repeat
  input "inventory number? ":number
 until number <= number'of'records
 read file 2, number:description$, quant
 ity, reorder'level, price // wrap line
 print "description: ";description$
 print "quantity on hand: ";quantity
 input "amount to add? ":add
 quantity:+add
 write file 2, number: description$, qua
 ntity, reorder'level, price//wrap line
 close file 2
endproc
```

The procedure for subtracting is identical except for the assignment statement.

```
proc subtract'from'stock
 open file 2, "inventory", random 80
 repeat
  input "inventory number? ":number
 until number <= number'of'records
 read file 2, number: description$, quan
 tity, reorder'level, price//wrap line
 print "description: ";description$
```

```
 print "quantity on hand: ";quantity
 input "amount to subtract? ":subtract
 quantity:-subtract
 write file 2, number: description$, qua
 ntity, reorder'level, price// wrap line
 close file 2
endproc
```

For listing the items that are below the reorder list, we'll do something slightly different. Let's have the selected records line up under a horizontal title in the form of a table. We'll print the table headings, then open the file and select the appropriate records whose quantity on hand falls below the reorder'level.

```
proc list'reorder
 zone 10 // column width
 open file 2, "inventory", random 80
 print
 print tab(3);"items to be reordered"
 print
 print "=============================="
 print "description    quantity   reorde
 r unit" // wrap line
 print "of item        on hand    level
   price" // wrap line
 print "=============================="
 for number:=1 to number'of'records do
  read file 2, number: description$, qua
  ntity, reorder'level, price//wrap line
  if quantity <= reorder'level then
   print "description$",
   print using "########":quantity,
   print using "########":reorder'level;
   print using "$####.##":price
  endif
 endfor number
 close file 2
 print
 print "end of listing.  press any key."
 while key$ = chr$(0) do null
 zone 0
endproc
```

Scrolling through the inventory could be a similar procedure without the if...then statement, or it could show one record at a time, letting the user bring up the next record by pressing the return key. Here's how the second option could be coded.

```
proc scroll
 open file 2, "inventory", random 80
 for number:= 1 to number'of'records do
  read file 2, number: description$, qua
  ntity, reorder'level, price//wrap line
  print "=============================="
  print "description: ";description$
  print "on hand:     ";quantity
  print "reorder at:  ";reorder'level
  print using "unit price $###.##":price
  print "=============================="
  input "hit return for next one":reply$
  print
 endfor number
 close file 2
 print
 print "press any key for menu"
 while key$ = chr$(0) do null
endproc
```

Nothing unusual there. Destroying a file and starting over will, hopefully, be used very seldom. An appropriate warning message will appear before the action occurs.

```
proc initialize
 input "Do you really want to destroy th
 e file (y/n)?":reply$ //wrap line
 if reply$="y" then delete "0:inventory"
endproc
```

That's it. A simple inventory program which you can modify for the number and complexitiy of records, formatting of output, etc. You might consider adding a printing option (select "lp:") for the output of one of the procedures. Since the inventory itself resides on the disk, its size is limited only by the amount of externally addressed memory. Simply adjust the number'of'records constant in the first procedure.

TUTORIAL DISK STRIKES BACK

or PROGRESS WITH COMAL

We found part of a letter from David Skinner very interesting. Here is what he says: " Yesterday, I met with two groups of students at Ouachita High School. COMAL received a big welcome. Today I met with the teachers and received a somewhat cooler reception. It seems that using the TUTORIAL DISK, I accidently covered all of their course material for the next six weeks. The teachers were simply not prepared for such rapid progress by the students.

POLAR GRAPHS WITH COMAL

by Ron France

For several years I have been teaching high school students to graph equations using polar coordinates. I have often wished to have a computer program which would display a detailed graph. COMAL version 0.14 for the Commodore 64 does this very nicely.

In rectangular coordinates, each point is plotted from a pair of numbers which indicate the horizontal and vertical distances from the origin. However in polar coordinates, a point is located when given both its distance and its direction from the origin, or **pole**. The direction is measured as an angle counterclockwise from the polar axis, a ray extending to the right of the pole. Thus the pair of numbers, $(4, 90^0)$ indicates a point four units above the pole.

Before you run the included program, you need to type the equation you wish to graph as a line of the program. (Inside FUNCtion F at the end of the program). In the program the variable r represents the distance from the origin and t represents the direction. Your equation must be in the form of r = function of t.

You can use this program to study various types of polar equations. For example an equation of the form

```
r = b + a*sin(t)
```

is called a **limacon**. Run the program with various values of a and b. For example try r = 2 + 2*sin(t) or r = 2 + 3*sin(t). What happens when b is greater than a? What happens when b = a? Or when b is less than a?

An equation in the form of

```
r = a*sin(n*t)
```

is called a **rose curve**. How many petals are on the rose when n = 2? When n = 3? Or when n = 4? Can you find the rule which tells how many petals there will be? If you change the value of a, what happens to the rose?

Here are a couple other forms to experiment with:

```
r = t/a              (a spiral)
r = b + a*sin(n*t)
```

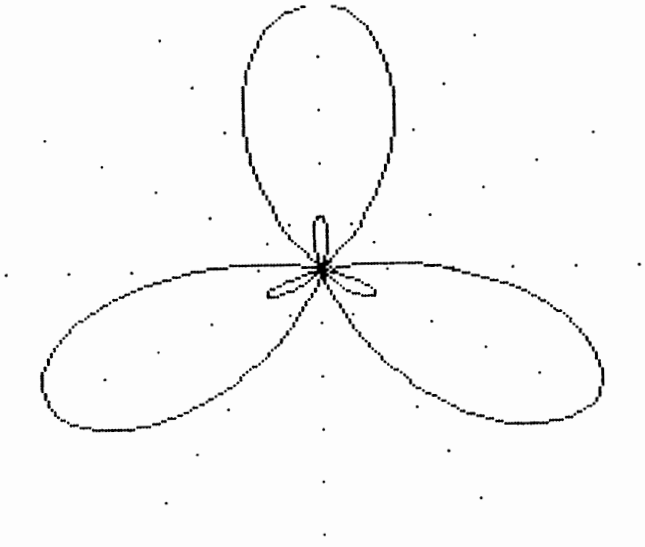
Also try these same forms but replace sin with cos.

In addition, you are able to change the shape and size of the graph by changing some of the values at the beginning of the program. XORG and YORG are the respective horizontal and vertical locations of the origin. HSCALE and VSCALE determine the respective horizontal and vertical scales (sizes) of the graph. I have found that it works best if VSCALE is 80% of HSCALE. INC represent the increment at which the points are plotted. The lower the value of INC, the more accurate the graph, although it takes longer to make. (On the disk is a longer version of this program which permits these values to be entered during the run of the program.)

When you run the program, you will first see some reference points being plotted in concentric circles. Each circle is one unit larger than the one before it. Points are plotted along each circle at thirty degree intervals. It is also instructive to watch the turtle. It is always pointing in the direction of t as t is incremented. You will be able to learn much about polar graphing from this program. Even if you are not acquainted with the mathematics involved, you will enjoy watching the variety of graphs being made.

```
// polar graphs
xorg:=160; yorg:=100
hscale:=25; vscale:=20
inc:=2
setgraphic 0
hideturtle
penup
background 15
border 12
pencolor 11
clear
penup
for r:=1 to 100/vscale do
 for dg:=0 to 360 step 30 do
  theta:=dg*.0174533
  find(r,theta)
```

```
  plot i,j
 endfor dg
endfor r
r:=f(0)
find(r,0)
drawto i,j
pendown
pencolor 6
showturtle
for dg:=0 to 360 step inc do
 setheading 90-dg
 theta:=dg*.0174533
 r:=f(theta)
 find(r,theta)
 drawto i,j
endfor dg
hideturtle
repeat
 x:=0
until x<>0
end
proc find(r,t)
 x:=r*cos(t)
 y:=r*sin(t)
 i:=hscale*x+xorg
 j:=vscale*y+yorg
endproc find
func f(t)
 // here is the equation ...
 // express r as a function of t
 r:=3*sin(8*t)+1
 return r
endfunc f
```



R = 2 - 3*SIN(3*T)

1520 PLOTTER 2.0 GRAPHICS DUMP

This program will dump a graphics screen to the 1520 plotter. Start it up and then go watch a movie - it takes about 2 hours.

```
proc print'screen closed
 // for 1520 plotter and comal 2.0
 use graphics
 open file 2,"u6:/s1"
 print file 2: "h"
 for y=199 to 0 step -1 do
  x=0
  print file 2:"m";x;chr$(29);y;chr$(29)
  repeat
   if getcolor(x,y)<>1 then
    print file 2: "d";x;chr$(29);y+1;chr
    $(29) // wrap line
   else
    print file 2: "m";x;chr$(29);y+1;chr
    $(29) // wrap line
   endif
   x:+1
  until x>319
 endfor y
 close file 2
endproc print'screen
```

GUESS IT INSTRUCTIONS

by Carl Frerichs

The rules are simple:

1) The computer asks you questions
2) You answer questions
3) The computer takes an educated guess at what you described with your answers
4) if it guesses wrong it asks you to more fully describe what you were thinking of and then reprograms itself so that you can't fool it a second time with the same thing.

In case you didn't read instruction (4) completely, I'll repeat something very important:

The computer reprograms itself with your information!

No joking around, honest! Everything the computer 'knows' is on DATA STATEMENTS in the program. These data statements consist of questions and answers which are linked together to allow the computer to give its educated guess. When you give the computer something new to think about it adds the new informaton and alters the way the old information is linked together. Then it will 'know' what you 'know'.

After you quit playing you may verify this for yourself by listing all the data statements from line 1000 to the end of the program. Your answers will be there. Also, your mispellings and mistakes. But, because your answetrs are actually data statements you could fix them.

SPECIAL COMMANDS

q=quit= you are finished and/ or you want to fix some spelling errors or just bad answers.
l=list= you want to see what the computer already 'knows'
s=save= save the latest and greatest GUESS IT game to disk so you don't loose it and so the computer will be much smarter the next time you play.
y=yes= acceptable affirmative response to a question asked by the computer, when applicable.
n=no= negative response when applicable.

SPECIAL NOTES

This version of the GUESS IT game starts with three basic catagories; Animal, vegatable, and Mineral.

The optimum results will be achieved if you place EVERYTHING in these three catagories. (ie, put fruits in the vegatable category; put machines in the mineral category; etc.)

Place only inmaterial items such as thoughts or energy or time outside of these three categories. Trust me. You can only get out of the game what you put in.

Besides, when you want to impress your friends with how smart your computer is compared to theirs, you'll want to be accurate and sensible.

For more ideas on using GUESS IT in other ways (ie, LEARNING MACHINE or same game different subject) see ADVANCED GUESS IT NOTES.

ADVANCED GUESS IT - Special Instructions

or THE TAO OF THE BINARY TREE

The structure of the DATA STATEMENTS in the GUESS IT game is at once ridgid and yet flexible. This gives it a potential much greater than the simple game of TWENTY QUESTIONS which is what GUESS IT was based on.

The easiest modification is simply a change of SUBJECT. Take for example the many battles of the US Civil War. For each there is a Decisive action in battle or, a Date and Time which had a significance beyond the normal scope of the battle or, the personal Victory or Death of a Key Figure or Figures. Let these be the questions and answers and you have a very specific History Drill. And it takes no time at all for an Instructuor to build the drill program as the program will build itself as the questions and answers are supplied to it!

Instead of asking: Animal, Vegatable, or Mineral; ask what the most significant means of identifying the battle concerned is, with respect ot its LOCATION, DATE & TIME, or INDIVIDUALS.

It's just that simple.

The subjects are whatever you can think of. The computer can 'learn' it, if you can ask and answer questions. For example take the following:

-English Grammer-
Parts of a Sentence,

-Chemistry-
Elements of the Periodic Table

-Geology-
Phenomena in the US National Parks

-Drama-
Characters of the Plays and Sonnets of William Shakespere

-Basic Math-
Determine a Whole Number from its Value Squared

As you can readily see, if you can ask a series of YES or NO questions and arrive at a single answer the choice of subject is unlimited.

Now, for another modificaton.

As the program stands now, the data statements are for the most part made up of questons(q) and answers(a). We can introduce a third kind of data statement which can be manually inserted in to the program and which would replace the answer data statement. Let's call this the CHAIN(c) data statement. Currently, in the procedure ASK'A'QUESTION only 'q' and 'a' data statements are processed. With a single test for a CHAIN(c) data statement we can choose to chain to a new program. The new program could be a continuation of the 'question & answer' process or, something entirely different, such as a specific tutorial for review of the material being covered. You say what good would this do? I said in the original basic GUESS IT notes that you only get out what you put in, and that you should be accurate and sensible. Now I can tell you that you should occasionally enter something which is incorrect or inaccurate!

That's right. Incorrect and inaccurate. How else can you really be sure someone isn't just 'punching keys' or playing instead of paying attention? So, a few well placed land mines that chain to a 'gatcha' program should help maintain someones attention.

Of course, with the 'chaining' feature you could have a program/ topic menu, but this could be somewhat tedious when the program/ topic you want is the last link in the data statement structure.

Speaking of the DATA STATEMENT STRUCTURE, it's time to quit selling and do some describing.

The first DATA STATEMENT, which must always be line 1000, contains the number of the next data statement to be built.

The second DATA STATEMENT must always contain the First and therefore most Primary question you wish to ask. Since it is a yes/ no question it will effectively eliminate 50% of the remaining questions.

The remaining DATA STATEMENTS may be answer(a) and question(q) data statements intermingled in any fashion.

To return to the structure of the 'q' data statement it is as follows:

\qxxxx\$yyyy\#zzzz\

where xxxx= a question
yyyy= a data statement line number from 1001 to 9999
zzzz= another line number
$= the letter 'Y' or 'N' but not the same as the value of #
#= the letter 'N' or 'Y' but not the same as the value of $.

Since each question has only two answers the system utilizing this technique is called BINARY. And a series of this type of questions ia called a BINARY TREE.

The structure of the 'a' Data Statement is as follows:

\axxxx

where xxxx= an answer linked to a specific questions YES or NO response

Final notes: (I bet you are glad I said that!)

I mentioned earlier that the first question iliminates 50% of the questions in the DATA STRUCTURE. That's the way the program builds the instructions. Too bad, sorry! But, wait. I also said that once you quit(q) the program, you could then edit the data statements to fix misspellings and other errors. So, guess what that means? Yes, that's right. You can change that old open ended ever learning structure into a closed or partially closed loop which allows:

1) questions and answers which would normally have been bypassed or at the very leas duplicated to be accessed from anyplace else in the structure.

2) a finite, closed structure can be built for tutorials, to force the participant to work within the constraints given by the instructor.

As an example to the last point I could have made the negative or 'N' response in the GUESS IT game, for the question 'IS IT MINERAL return to the first question. This would force all answer to be in the catagories of Animal, Vegatable, or Mineral. No exceptions for Ideas or Energy or anything else.

For more information on how the program manipulates existing data statements adn builds new statements to a running program see COMAL LANGUAGE NOTES.

COMAL LANGUAGE NOTES

As with BASIC some MAIN MEMORY locations are not estabished until the actual running or a COMAL program. Two of these, which are used effectively in the GUESS IT program are locations 251 and 252. These contain the LO and HI portions of the address of the next DATA STATEMENT to be read after the execution of the first READ instruction.

```
Loc=peek(251)+256*peek(252)
```

Therefore, if the values in 251 and 252 are saved for a particular DATA STATEMENT, before it is read, you can return directly to that same data statement ready to once again perform a READ by restoring the saved values into 251 and 252. This is called a MARK and RESET operation or a Controlled RESTORE, whichever terminology you prefer.

As to the internal structure of a 'simple' (single item) DATA STATEMENT it is as follows:

offset	desc.
-2=	Addr Low
-1=	Addr High
0=	Line# High
1=	Line# Low
2=	Line Lth
3=	DATA Cmd
4=	Seperator
5=	Data Lth
6=	Data Item
7=	Terminator
8=	Next Cmd Addr Low
9=	Next Cmd Addr High

Where 0= the byte which the value contained in 'Loc' is addressing or, if you prefer, pointing at.

I hope some other command enthusiasts will find this informative. I also hope that people who read the COMAL MEMORY MAP and see bytes listed as 'Unused or Free Memory' will not rush out ans start storing little bits of information here and there! They may be surprised at how their program runs, if at all. I would recommend the phrase 'UNKNOWN USE' for undocumented bytes of memory, especially in the Page Zero Area.

For an example of the values in 251 and 252 being put to use play the game **GUESS IT** and watch DATA STATEMENTS being dynamically changed by a running program.

COMAL 2.0 LIGHT PEN USE

by Bob Schulz

The new COMAL cartridge now makes it incredibly easy to include a lightpen in your programs for the Commodore 64. The built-in commands were designed to work with the Madison Computer "McPen" which is the pen I used. Both the COMAL Cartridge and the "McPen" lightpen are available from the COMAL Users Group, USA, Limited, 5501 Groveland Terrace, Madison, WI 53716. In a few lines of code, a drawing or menu program can be quickly written.

The lightpen package is activated by the statement "USE LIGHTPEN". This adds the following commands to COMAL:

```
READPEN(<x coord>,<y coord>,<penon?>)
ACCURACY(<x range>,<y range>)
DELAY(<time>)
OFFSET(<x correction>,<y correction>)
TIMEON(<time>)
PENON
```

The x and y values received from the commands are generally positive (depending on the OFFSET numbers you have given the pen). On a pen with properly adjusted OFFSET, x and y equal zero in the bottom left hand corner of the screen. X varies up to 355 at the right hand side and y varies up to 200 at the top. Each pixel in graphicscreen(0) has its unique address.

Basically the commands operate like this: The READPEN command returns an x and y coordinate which may be viable if the pen is close enough to the screen (i.e. the value of penon returned = 1 (true)). This value is accurate to a range which you specify in the ACCURACY command. The value is not returned until the same value has been read by the pen for the number of times that you specify in the DELAY command. The value returned is corrected by the OFFSET numbers you have prescribed in the OFFSET command. If the pen is moved away from the screen, <penon?> will remain true for a <time> set in the TIMEON command. The ACCURACY, DELAY, OFFSET and TIMEON instructions need not be included in the program, but if they are not, the default values are assumed. (More about this later.) READPEN gives you the <penon?> value, but that value alone will be returned when the PENON command is invoked. (An extended explaination of these commands is given in The Amazing Adventures of Captain Comal - Cartridge Graphics and Sound, part of the Cartridge Support Package).

A drawing program can be as simple as:

```
use lightpen
use graphics
graphicscreen(0)
REPEAT
 readpen(x,y,p)
 IF p=1 THEN drawto(x,y)
UNTIL key$<>chr$(0)
textscreen
```

This program will draw until any key is pressed.

To improve the program, the OFFSET command could be used to insure that the line is drawn to a point exactly under the pen. By changing the accuracy to higher values, the time required to plot a point can be reduced without a noticeable reduction in sketching capability. There does seen to be some tendency of the "McPen" to give erroneous readings in spite of the accuracy. Since these values were for x>355 I simply eliminated the problem by ignoring values of x larger than 355.

As an aid in writing menu programs I suggest inserting the code below after the menu printing routine:

```
REPEAT
 PRINT AT 20,1:"              "
 readpen(x,y,p)
 PRINT AT 20,1:x;y
UNTIL true=false
```

Next run the program, and the x,y, coordinates of the pen will be displayed in the lower left hand corner of your menu. Move the pen to the location on menu that is intended to choose a particular action. Now it is possible to move the pen over the area and the x,y values will be displayed. It is then easy to see what range of x and y values will be acceptable for a given menu selection. For example, if your menu printing program looked like this:

```
PRINT AT 5,5:"What did you eat?"
PRINT AT 10,5:"Fish                  *"
PRINT AT 14,5:"Something Edible      *"
```

then beginning at the asterisk, move the pen up, down, left and right as far as you are willing to allow an uncoordinated individual to wonder and still choose that particular menu action. Then record the x and y values. For this example a selection of fish should be 220<x<260 and 110<y<130. A selection of "Something Edible" would have the same x values, but a verticle range of 80<y<100. The rest of the program might look like this:

```
readpen(x,y,p)
IF p=1 AND x>220 AND x<260  THEN
 IF y>110 AND y<130 THEN
  PRINT AT 15,5:"YUK !                "
 ELIF y>80 AND y<100 THEN
  PRINT AT 15,5:"Fish, unfortunately.."
 ELSE
  PRINT AT 15,5:"Move pen to * wanted "
 ENDIF
ELSE
 PRINT AT 15,5:"Move pen to * wanted "
ENDIF
```

I have noticed that the pen senses light up to 4 inches from the screen. This causes "p" to be set equal to 1 and hence "viable" x and y values are expected by the program. Unfortunately at 4 inches from the screen it is unlikely that the pen is pointing at the intended location on the screen. This could cause an incorrect menu selection. It is possible then to use the DELAY command to slow down the return of the READPEN values. A DELAY value of between 10 and 20 will probably work well depending on your needs.

I had a problem figuring out the default values for the command parameters. Here are the ballpark values that I came up with.

```
ACCURACY(7,7)
DELAY(15)
OFFSET(75,-40)
TIMEON(20)
```

It seemed odd to me at first that a nonzero value of TIMEON was used, but the pen does not sense light as well when it is being moved. It is possible to draw faster when the pen is held on with the TIMEON command because it does not allow the pen to think that it has been pulled away from the screen. In applications where a touch of the pen to the screen does something TIMEON must be given a low value or the action will continue after the pen has been pulled away.

You may notice the pen becoming less sensitive after a few hours of use. Evidently the plastic lens on the phototransistor becomes static charged from the screen, and then collects dust. Clean it periodically with a damp cloth.

Those individuals who wish to do accurate drawings will be disappointed with this pen. The best accuracy I could get was about +/- 4 pixels in the x direction and +/- 2 pixels in the y direction. However, for games, menu selection, and quick sketching it works well. My children thought it was the best thing since ice cream, because unlike a keyboard, joystick or paddle, the operation is both intuitive and magical. I feel the "McPen" is a good buy for the money and the COMAL 2.0 Cartridge is the only way to use it!

TODAY DISK 1

```
"boot c64 comal"
"c64 comal 0.14"
"----------------"
">--------------<"
">error messages<"
">    file      <"
">------------ -<"
"comalerrors"
">---------- --<"
">file generator<"
">---------- ---<"
"generate errfile"
">--------- ----<"
">auto boot prog<"
">-------- -----<"
"hi"
">------- ------<"
">seq data files<"
">------ -------<"
"information84mar"
"help-comal"
"help-graphics"
"help-sprites"
">----- --------<"
">comal programs<"
">---- ---------<"
"see'information"
"see'instructions"
"°°°°°°°°°°°°°°°°"
"big'letter.l"
"big'letter"
"bigletter/demo.l"
"bigletter/demo"
"logo'emulator.l"
"logo'emulator"
"dir'lister.l"
"dir'lister"
"sky'catcher.l"
"sky'catcher"
"create'lander.l"
"create'lander"
"lander'sprites"
"lander.l"
"lander"
"microscribble.l"
"microscribble"
"paddletest.l"
"paddletest"
"etchasketch.l"
"etchasketch"
"light'pen.l"
"light'pen"
"koala.l"
"koala"
"color'funcs.l"
"color'combo.l"
"color'combo"
"hidescreen.l"
"showscreen.l"
"print'using.l"
"print'using"
"note17.program"
"val/demo.l"
"val/demo"
"square-a3"
"square-b3"
"design3"
"design4"
"square-c3"
"squares3"
"sprite-a1"
"sprite-b1"
"link_program.l"
"link_program"
"control.l"
"new'program.l"
">--- ----------<"
">the following <"
">two programs  <"
">are written in<"
">   basic      <"
">   -----      <"
">do not load   <"
">them into     <"
">comal.        <"
"> ------------ <"
"1541backup(free)"
"single file copy"
```

TODAY DISK 2

```
"boot c64 comal"
"c64 comal 0.14"
"----------------"
">--------------<"
">error messages<"
">    file      <"
">------------ -<"
"comalerrors"
">---------- --<"
">file generator<"
">---------- ---<"
"generate errfile"
">--------- ----<"
">auto boot prog<"
">-------- -----<"
"hi"
">------- ------<"
">seq data files<"
">------ -------<"
"information84mar"
"help-comal"
"help-graphics"
"help-sprites"
">----- --------<"
">comal programs<"
">---- ---------<"
"see'information"
"see'instructions"
"°°°°°°°°°°°°°°°°"
"8023p'options4"
"8023p'options4.l"
"background.l"
"benchmark64.basl"
"benchmark64.comal"
"benchmark64.l"
"border'color.l"
"clear'collisn.l"
"convert"
"convert.l"
"disk'get2.l"
"func3.l"
"inside.l"
"meeting"
"meeting.l"
"note76.l"
"note79.l"
"note80-1.l"
"note80-2.l"
"note81-1.l"
"note81-2.l"
"note96.l"
"note97.l"
"page 15 sample.l"
"page.l"
"pens'color.l"
"pentagram.l"
"plottext.l"
"proc1.l"
"proc2.l"
"put'char.l"
"reverse.l"
"screen'char.l"
"set'screen.l"
"sinename"
"sinename.l"
"front'page"
"sprite'editor21"
"stars1.l"
"stars2.l"
"string"
"string.l"
"vt-52.v4"
"vt-52.v4.l"
"plexpac.l"
"vecpac.l"
"matpac.l"
"rotpac.l"
"test lsf.l"
"test'clog.l"
"test'cp'r.l"
">--- ----------<"
">the following <"
">two programs  <"
">are written in<"
">   basic      <"
">   -----      <"
">do not load   <"
">them into     <"
">comal.        <"
"> ------------ <"
"1541backup(free)"
"single file copy"
```

TODAY DISK 3

```
"boot c64 comal"
"c64 comal 0.14"
"7777777777777777"
",99999999999999;"
"6error messages5"
"6    file      5"
"<88888888888888>"
"comalerrors"
",99999999999999;"
"6file generator5"
"<88888888888888>"
"generate errfile"
",99999999999999;"
"6auto boot prog5"
"<88888888888888>"
"hi"
",99999999999999;"
"6seq data files5"
"<88888888888888>"
"information84jun"
"help-comal"
"help-graphics"
"help-sprites"
",99999999999999;"
"6comal programs5"
"<88888888888888>"
"bit'map'print.l"
"circle.l"
"clock"
"comal'dump.asm"
"comal'dump.bas"
"comal'dump.obj"
"convert(new).l"
"create.l"
"curcol.l"
"currow.l"
"cursor.l"
"demo"
"dir'manipulator"
"drawletter.l"
"getbackground.l"
"getborder.l"
"getpen.l"
"getpencolor.l"
"getspritecolor.l"
"getturtlesize.l"
"graphicstate.l"
"heading.l"
"hidescreen.l"
"hidescreen2.l"
"jiffies.l"
"keyword'print"
"load'demo"
"load'demo2"
"load'screen.l"
"load'screen2.l"
"moire.hrg"
"mount.l"
"obj'load.l"
"obj'save.l"
"page.l"
"penstate.l"
"pie'chart'maker"
"pie'chart'print"
"plot'char.l"
"plot'char/demo"
"polygon.l"
"quicksort'number"
"quicksort'string"
"random'file/demo"
"randomize1.l"
"randomize2.l"
"save'demo"
"save'screen.l"
"set/readtime.l"
"setx.l"
"sety.l"
"showscreen.l"
"showscreen2.l"
"showsprite.l"
"spritestate.l"
"spritexcor.l"
"spritexsize.l"
"spriteycor.l"
"spriteysize.l"
"turtlestate.l"
"user'error/demo"
"xcor.l"
"ycor.l"
",99999999999999;"
"6the following 5"
"6basic program 5"
"6is a single   5"
"6drive copier. 5"
"????????????????;
"6do not load it5"
"6while in comal5"
"<88888888888888>"
"1541backup(free)"
```

TODAY DISK 4

```
"boot c64 comal"
"c64 comal 0.14"
"comalerrors"
"hi"
"abc.sprite"
"alpha1'gen"
"alpha1.dat"
"alpha2'gen"
"alpha2.dat"
```

"comal'bug-a"
"comal'bug-b"
"dec'to'hex/demo"
"diamond"
"disk'protector"
"disk'talk'examp1"
"disk'talk'examp2"
"dodge'em"
"dump'1525"
"dump'big'epson.l"
"dump'nec8023a"
"dump'prowriter"
"dumpscreen.proc"
"dumpscreen/demo"
"dumptext1525.l"
"dynam'data"
"find'string/demo"
"gutenberg'shell"
"gutenberg'Ademo"
"gutenberg'Bdemo"
"hex'to'dec/demo"
"joystick.proc"
"lock'lower.proc"
"lock'upper.proc"
"logical'ops.func"
"ml'string/demo"
"music'all/demo"
"num-string/demo"
"opt'triangle"
"optical'hexagon"
"paddle.proc"
"polyspirals"
"prime/demo"
"ram'errors"
"random'plot"
"rocket.sprite"
"screen'location"
"scroll'down/demo"
"seq'print"
"spirolateral"
"stars"
"takeoff/demo"
"tiles"
"tri'hex/demo"
"two'drive'copier"
"two'drive'instru"
"unlock'case.proc"
"wall'clock"
"wandering"
">--- ----------<"
">the following <"
"> program is <"
"> written in <"
"> basic <"
"> ------------ <"
"1541backup(free)"

TODAY DISK 5

"hi"
"comalerrors"
">comal programs<"
"alarm'system"
"arc'trig'functns"
"boxtree"
"checkbook"
"color'mix"
"connected'boxes"
"correlator"
"datacollision"
"disk'editor"
"dog/cat"
"draw'sine'wave"
"exmpl'bar'graph"
"expand'memory"
"grade'book'exmpl"
"grade'distribute"
"hypotenuse"
"identify7"
"inventoryprogram"
"leap'year"
"let&using'exmpl"
"magic'fruit"
"many'patterns"
"many'stars"
"ml'setup"
"names'printout"
"okl92'hi"
"okl92'screen'io"
"pitfall'harry"
"polar'daisy"
"polar'long"
"queens(ver 0.14)"
"recursion'exmpl"
"rotated'ovals"
"show-stopper"
"sign'language"
"spritecollision"
"turtle/demo"
"wage'demo"
"xploded'pie"
">--data files--<"
"bigdump'nec.obj"
"bigdump'nec.src"
"dat.bwv783"
"fingera"
"fingerb"
"fingerc"
"fingerd"
"fingere"
"fingerf"
"fingerg"
"fingerh"
"fingeri"
"fingerj"
"fingerk"
"fingerl"
"fingerm"
"fingern"
"fingero"
"fingerp"
"fingerq"
"fingerr"
"fingers"
"fingert"
"fingeru"
"fingerv"
"fingerw"
"fingerx"
"fingery"
"fingerz"
"inventory"
"okl92.dump.obj"
">---listings---<"
"base'convert.l"
"clear'keys.proc"
"func.modemget2.0"
"graphs.l"
"ml'procs"
"save'screen.proc"
"test'signal.l"
">---screens----<"
"1st 80 kanji.hrg"
"2nd 80 kanji.hrg"
">-2.0 programs-<"
"all'at'once2"
"memory'peeker2.0"
">--benchmarks--<"
"sieve.comal"
"sieve.basic"

USER DISK 1

"boot c64 comal"
"c64 comal 0.14"
"comalerrors"
"hi"
">------- ------<"
"--SCREEN DUMP---"
"1525 screen dump"
"--DISK COPIERS--"
"sd2 copier"
"sd2'copy&label"
"--LABEL MAKERS--"
"label"
"mail'label'hs"
"---PROCEDURES---"
"title'page.proc"
"shell'sort$.proc"
"header.proc"
"-PRINTER UTILITY"
"seq'print"
"-DEMO PROGRAMS--"
"3d'cube"
"aprilfool"
"beeper"
"bounce"
"c'curve"
"circles'kev"
"clock"
"cones"
"curve"
"design 2"
"design 3"
"design 4"
"diamond"
"dragon"
"eight boxes"
"etch-a-sketch"
"graf1"
"graf2"
"graf3"
"graf4"
"graf5"
"graf6"
"grid"
"hanoi"
"hilbert"
"life"
"p'circle"
"percent gain"
"rnd'bounce"
"serpent"
"spirolateral"
"spiro.plain"
"starry night"
"the'thing"
"turtlestick 6"
"turtlestick 7"
"who"

USER DISK 2

"boot c64 comal"
"c64 comal 0.14"
"comalerrors"
"hi"
"-FEATURE PROGRAM"
"gutenberg'shell"
"alpha1'gen"
"alpha1.dat"
"alpha2'gen"
"alpha2.dat"
"gutenberg'Ademo"
"gutenberg'Bdemo"
"--SCREEN DUMPS--"
"bit'print'epson"
"1525 pixel dump"
"---PROCEDURES---"
"x-ycor.proc"
"heading.proc"
"circle.proc"
"title'page.proc"
"----PROGRAMS----"
"basic'to'comal"
"coin flip"
"comal art"
"ct.header"
"curvette"
"find'string.demo"
"flowers"
"four circle"
"hamburger"
"hourglass"
"lifer"
"lissajous"
"new'house"
"nh3.exc"
"pinwheel"
"polymusic"
"spiral'star"
"squiral mod"
"star'power"
"starwatch"
"tabby"

USER DISK 3

"boot c64 comal"
"c64 comal 0.14"
"comalerrors"
"hi"
"-TWIN 1541 COPY-"
"tddcfc"
"tddcfc.inst"
"--SCREEN DUMPS--"
"prowriterdump.l"
"nec8023 scrndump"
"nec ml dump"
"ml'dump.obj"
"---PROCEDURES---"
"colors.proc"
"--SPRITE FILES--"
"abc.sprite"
"rocket.sprite"
"---UTILITIES----"
"dynam-data-8"
"84expenditures"
"-DEMO PROGRAMS-"
"a maz'in basic.b"
"a maz'in comal.c"
"a maz'in simon.s"
"angry.dragon"
"another'moire"
"atari graphics"
"bagel"
"beep/gong.demo"
"chris'star"
"circles"
"color swirl"
"comal promo"
"crazy quilt"
"fft'model"
"key draw"
"music'all'.demo"
"pascalstrekant"
"polar"
"rnd color sqs"
"shapes"

"sin'on'the'side"
"spiral'cir"
"spirograph"
"spotty"
"tiles"

USER DISK 4

"boot c64 comal"
"c64 comal 0.14"
"comalerrors"
"hi"
"----- ----------"
"--APPLICATIONS--"
"----PROGRAMS----"
"record keeper"
"label-ab"
"budget"
"address"
"---UTILITIES----"
"dumpscreen.1525"
"terminal"
"comalerror'gen"
"---PROCEDURES---"
"accept'demo"
"hersh demos"
"str$.proc"
"val2.proc"
"-DEMO PROGRAMS-"
"arabesque1"
"arabesque2"
"arabesque3"
"arabesque4"
"arabesque5"
"arabesque6"
"arabesque7"
"checkerboard"
"clue"
"comal 2.02 boot"
"dizzy turtle"
"drum"
"fanfare"
"flasher"
"keno.game"
"kenoboard"
"name game"
"perm'game"
"print boxes"
"rectan"
"rectangulus"
"scale"
"son of moire"
"son'of'pinwheel"
"spiral'sqr"
"sprite'circle"
"starwars"
"supersketch"
"weirdness"
"window"

USER DISK 5

"boot c64 comal"
"c64 comal 0.14"
"comalerrors"
"hi"
"--SCREEN DUMPS--"
"neccomaldump"
"txt.screendump.1"
"---FUNCTIONS----"
"asc.func"
"scr2petsci.func"
"---DATA FILES---"
"design.dat"
"flake.dat"
"e.dat"
"f.dat"
"g.dat"
"-DEMO PROGRAMS-"
"arabesque8"
"arcs.demo"
"card'dealer"
"circle'maker"
"circle2.demo"
"city'scape"
"color mix"
"color'pic'loader"
"cube designer"
"dog/cat"
"dot.and.line"
"drawshape.demo"
"drumette"
"envelope'print"
"fast'circle"
"fillbox"
"flurry"
"fourier sq"
"graphics demo"
"haiku"
"house"
"kaleidoscope"
"moon phase"
"moving'turtle"
"music'synth'eric"
"pauls math'add"
"poly'circles"
"polygoncrazy"
"probability"
"random'show"
"readsprite/demo"
"strs retirement"
"therapy"
"turtlestick8"
">LOAD FROM BASIC"
"midwesterm 4.0"

USER DISK 6

"boot c64 comal"
"c64 comal 0.14"
"comalerrors"
"hi"
"----FEATURED----"
"----PROGRAM-----"
"game'unfin"
"---DATA FILES---"
"--DO NOT LOAD---"
"lakeside"
"e"
"f"
"g"
"tbsk"
"cmk"
"kbit"
"farmbit"
"farmtbs"
"farmcm"
"treebit"
"treetbs"
"treecm"
"help1"
"cbit"
"ctbs"
"ccm"
"help2"
"---PROCEDURES---"
"beep.proc"
"cat'.proc"
"tod.proc"
"wait'n'go.proc"
"cursor.proc"
"----UTILITY-----"
"----PROGRAMS----"
"dir'print'nec"
"imp.dump"
"pretty printer"
"disk'editor"
"dual epson.dump"
"expand'comal"
"-DEMO PROGRAMS-"
"city patterns"
"comaldice"
"flower'judy"
"fractal"
"galactic news"
"orbit'circle"
"pitfall harry"
"polyhedron"
"queens"
"random keyframes"
"sphere'plot"
"traffic.light"
"variable boxtree"
"yarn'art1"

HANDBOOK DISK

"boot c64 comal"
"c64 comal 0.14"
"----------"
">----------<"
">error messages<"
">   file   <"
">---------- -<"
"comalerrors"
">---------- --<"
">file generator<"
">---------- ---<"
"generate errfile"
">--------- ----<"
">auto boot prog<"
">-------- -----<"
"hi"
">------- ------<"
">seq data files<"
">------ -------<"
"information84mar"
"help-comal"
"help-graphics"
"help-sprites"
">----- --------<"
">comal handbook<"
">procs & funcs <"
">use enter <"
">---- ---------<"
"create.1"
"cursor.1"
"even.1"
"fetch.1"
"file'exists.1"
"get'char.1"
"get'valid.1"
"jiffies.1"
"lower'to'upper.1"
"mount.1"
"new'employee.1"
"odd.1"
"pos.1"
"put'record.1"
"randomize.1"
"round.1"
"shift.1"
"control.1"
"commodore'key.1"
"take'in.1"
"value.1"
">- ---------- -<"
">sample progrms<"
">use load <"
">-- -------- --<"
"abs"
"and"
"append"
"atn"
"case"
"chr$"
"close"
"closed"
"cos"
"data"
"delete"
"dim"
"div"
"elif"
"else"
"end"
"endcase"
"endfor"
"endfunc"
"endif"
"endproc"
"endwhile"
"eod"
"eof"
"esc"
"exec"
"exp"
"false"
"file"
"for"
"func"
"if"
"in"
"input"
"int"
"key$"
"len"
"let"
"log"
"mod"
"not"
"null"
"of"
"open"
"or"
"ord"
"otherwise"
"pass"
"print"
"printfile"
"proc"
"random"
"read"
"ref"
"repeat"
"restore"
"return"
"rnd"
"select"
"sgn"
"sin"
"sqr"
"step"
"tab"
"tan"
"then"
"to"
"trap"

```
"true"
"until"
"using"
"when"
"while"
"write"
"zone"
">- - - - - - - <"
">bonus programs<"
"> - - - - - - -<"
"print'directory"
"utilities"
"file'to'print"
"file'to'screen"
"cursor/demo"
"value/demo"
"shift/demo"
"jiffy/demo"
"quicksort/demo"
"joystick/demo"
"paddle/demo"
"disk'get/demo"
"logical'ops/demo"
```

BEGINNING COMAL

```
"boot c64 comal"
"c64 comal 0.14"
">--------------<"
">error messages<"
">------------ -<"
"comalerrors"
">----------- --<"
">file generator<"
">---------- ---<"
"generate errfile"
">--------- ----<"
">auto boot prog<"
">-------- -----<"
"hi"
">------- ------<"
">seq data files<"
">------ -------<"
"information84mar"
"help-comal"
"help-graphics"
"help-sprites"
">----- --------<"
"exe17"
"exe18"
"exe19"
"exe110"
"exe22"
"exe23"
"exe31"
"exe32"
"exe34"
"exe36"
"exe41"
"exe42"
"discount"
"exe45"
"exe46"
"exe51"
"exe52"
"exe53"
"oddeven"
"exe71"
"exe72"
"exe73"
"peanuts"
"exe69"
"exe73b"
"bignuts"
"exe82"
"exe91"
"exe94"
"exe101"
"exe105"
"addition"
"exe112"
"unclexmas"
"exe122"
"exe123"
"exe131"
"exe141"
"test141"
"auntie"
"festivals"
"doctor"
"eggs"
"wheel"
"enrol"
"correct"
"entermarks"
"report"
"hannibal"
"wordgame"
"xmascards"
"xmasfile"
"enternames"
"enteroptions"
"newshop"
"writeoffer"
"addrpgr"
"exhibitoptions"
"> ------------ <"
"> data files   <"
">- ---------- -<"
"markbooks"
"xmas"
"options"
"offer"
"addresses"
">--- ----------<"
">   end of     <"
">beginningcomal<"
">                <"
">the following <"
">two programs  <"
">are written in<"
">   basic       <"
">   -----       <"
">do not load    <"
">them into      <"
">comal.         <"
">-- -------- --<"
"1541backup(free)"
"single file copy"
```

FOUNDATIONS

```
"boot c64 comal"
"c64 comal 0.14"
">--------------<"
">error messages<"
">------------ -<"
"comalerrors"
">----------- --<"
">file generator<"
">---------- ---<"
"generate errfile"
">--------- ----<"
">auto boot prog<"
">-------- -----<"
"hi"
">------- ------<"
">seq data files<"
">------ -------<"
"information84mar"
"help-comal"
"help-graphics"
"help-sprites"
">----- --------<"
"program1"
"program2"
"program3"
"program4"
"program5"
"program6"
"program7"
"program8"
"program9"
"program10"
"program11"
"program12"
"program13"
"program14"
"program15"
"program16"
"program17"
"program18"
"program19"
"program20"
"program21"
"program22"
"program23"
"program24"
"program25"
"program26"
"program27"
"program28"
"program29"
"program30"
"program31"
"program32"
"program33"
"program34"
"program35"
"program36"
"program37"
"program38"
"program39"
"program40"
"program41"
"program42"
"program43"
"program44"
"program45"
"program46"
"program47"
"program48"
"program49"
"program50"
"program51"
"program52"
"program53"
"program54"
"program55"
"program56"
"program57"
"program58"
"program59"
"program60"
"program61"
"program62"
"program63"
"program64"
"program65"
"program66"
"program67"
"program68"
"program69"
"program70"
"program71"
"program72"
"program73"
"program74"
"program75"
"program76"
"program77"
"program78"
"program79"
"program80"
"program81"
"program82"
"program83"
"program84"
"program85"
"program86"
"program87"
"program88"
"program89"
"program90"
"program91"
"program92"
"program93"
"program94"
"program95"
"program96"
"program97"
"program98"
"program99"
"program100"
"program101"
"program102"
"program103"
"program104"
"program105"
"program106"
"program107"
"program108"
"program109"
"program110"
"program111"
"program112"
"program113"
"program114"
"program115"
"program116"
"program117"
"program118"
"> ------------ <"
"1541backup(free)"
"single file copy"
```

STRUCTURED PROG

```
"boot c64 comal"
"c64 comal 0.14"
"comalerrors"
"hi"
"information84mar"
"help-comal"
"help-graphics"
"help-sprites"
">----- --------<"
"sec1.5"
"sec1.7"
"sec1.7b"
"prob1.1"
"prob1.2"
"prob1.3"
"prob1.4"
"prob1.5"
"sec2.5.1"
```

```
"prob2.11"
"prob3.3"
"cursor.l"
"sec3.2"
"sec3.2b"
"sec3.2c"
"example3.3"
"example3.4"
"example3.5"
"example3.6"
"example3.6b"
"example4.2"
"example4.3"
"example4.3b"
"example4.5"
"sec4.4"
"example4.6"
"sec4.5"
"example4.7"
"example4.8"
"sec5.1"
"example5.1"
"sec5.2.2"
"sec5.2.3"
"sec5.2.3b"
"example5.2"
"charcount.l"
"example8.1"
"example8.2"
"example8.3"
"example8.3b"
"fig8.1"
"sec8.4.1"
"sec8.5"
"sec8.5b"
"sec10.2"
"sec10.3"
"sec10.4.1"
"sec10.4.1b"
"sec10.4.1c"
"sec10.4.2"
"quicksort.l"
"example11.1"
"example11.2"
"example11.3"
"example11.4"
"example11.5"
"example11.6"
"sec12.3.1-write"
"sec12.3.1-read"
"sec12.3.1b-write"
"sec12.3.1b-read"
"sec12.3.1c-read"
"setfile.l"
"searchfile.l"
"validate.l"
"sec12.4-write"
"sec12.4-read"
"sec12.4-both"
"example12.3"
"example12.4"
"example12.5"
"solution3.4"
"solution3.5"
"solution3.6"
"solution3.7"
"solution3.8"
"solution3.9"
"solution3.10"
"solution3.11"
"solution3.12"
"solution3.13"
"solution3.14"
"solution3.15"
"solution3.17"
"solution3.18"
"solution3.19"
"solution3.20"
"solution3.21"
"solution3.22"
"solution3.23"
"solution3.24"
"solution3.25"
"solution4.3"
"solution4.4"
"solution4.5"
"solution4.6"
"solution4.7"
"solution4.8"
"solution4.9"
"solution4.10"
"solution4.11"
"solution4.12"
"solution4.13"
"solution4.14"
"solution4.15"
"solution4.16"
"solution4.17"
"solution4.18"
"solution4.19"
"solution4.20"
"solution4.21"
"solution5.1"
"solution5.2"
"solution5.3"
"solution5.4a"
"solution5.4b"
"roulette.l"
"solution8.1"
"solution8.2"
"solution8.3"
"solution8.4a"
"solution8.4b"
"solution8.5"
"solution8.6"
"solution8.7"
"solution8.8"
"solution8.9"
"solution8.10"
"solution8.11"
"solution8.12"
"solution8.13"
"solution8.14"
"solution8.15"
"> ------------ <"
"1541backup(free)"
```

GET ORGANIZED

```
"boot c64 comal"
"c64 comal 0.14"
"comalerrors"
"generate errfile"
"hi"
"information84jun"
"help-comal"
"help-graphics"
"help-sprites"
"help-instruction"
"°°°°°°°°°°°°°°°°"
"compare'dir"
"delete'dir"
"disk'summary"
"dos'menu"
"find'file"
"master'maker"
"print'dir"
"print'ids"
"startup"
"update"
"view'dir"
">- ---------- -<"
">procs follow: <"
">-- -------- --<"
"choices.l"
"disk'get.l"
"dual'drive.l"
"file'exists.l"
"get'dir.l"
"init.l"
"intro.l"
"menu.l"
"menu2.l"
"menu3.l"
"page.l"
"print'dir'reg.l"
"print'dir'label.l"
"printer.l"
"quicksort.l"
"read'dir'part2.l"
"read'dir.l"
"read'dir2.l"
"screen.l"
"see.l"
"set'updated.l"
"sort'ids.l"
"type'of'dir.l"
"verified.l"
">--- ----------<"
">end of comal  <"
">              <"
">the following <"
">two programs  <"
">are written in<"
">   basic      <"
">   -----      <"
">do not load   <"
">them into     <"
">comal.        <"
"> ------------ <"
"1541backup(free)"
"single file copy"
```

PROC & FUNC

```
"bootcomal.bas"
"c64 comal 0.14"
"comalerrors"
"hi"
"copy files.bas"
"control.l"
"link'program"
"accept.l"
"acos.l"
"arc'as.l"
"arc.l"
"arcl.l"
"arcr.l"
"ascii.l"
"asin.l"
"aspect.l"
"back'color.l"
"bin'str.l"
"bin.l"
"bit'ml.l"
"bitand.l"
"bitor.l"
"bitxor.l"
"border'color.l"
"bubsort'str.l"
"bubsort.l"
"cen.l"
"circle'as.l"
"circle.l"
"clearcoll.l"
"convert.l"
"cot.l"
"create.l"
"csc.l"
"curcol.l"
"currow.l"
"cursor.l"
"date.l"
"day'of'week.l"
"day'of'year.l"
"deg.l"
"delay.l"
"disk'get.l"
"draw.l"
"easter.l"
"eps.l"
"expb.l"
"fah.l"
"fp.l"
"get.l"
"getspritecolor.l"
"graphicstate.l"
"heading.l"
"hex'str.l"
"hex.l"
"inf.l"
"inkey.l"
"insertsort'str.l"
"insertsort.l"
"ip.l"
"jiffies.l"
"joystick.l"
"koala.l"
"leap'year.l"
"lightpen.l"
"load'screen.l"
"logb.l"
"lpad.l"
"ltrm.l"
"lwrc.l"
"matadd.l"
"match'r.l"
"matequal.l"
"matinvert.l"
"matmult.l"
"matscale.l"
"mattransp.l"
"matunit.l"
"matvec.l"
"max.l"
"maxlen.l"
"metric.l"
"min.l"
"mount.l"
"move.l"
"obj'load.l"
"obj'save.l"
"paddle.l"
"page.l"
"pen'color.l"
"penstate.l"
"pi.l"
"play.l"
"plot'char.l"
"poly'as.l"
"poly.l"
"print'at.l"
"put'char.l"
"quicksort'str.l"
"quicksort.l"
"rad.l"
"randomize1.l"
```

```
"randomize2.l"
"rect.l"
"remain.l"
"replace'char.l"
"replace'str.l"
"reverse.l"
"rpad.l"
"rtrm.l"
"save'screen.l"
"screen'char.l"
"sec.l"
"select'lc.l"
"set'time.l"
"settime.l"
"setx.l"
"sety.l"
"shellsort'str.l"
"shellsort.l"
"shift'wait.l"
"showsprite.l"
"shuffle'str.l"
"shuffle.l"
"sid.l"
"sin2.l"
"spritestate.l"
"spritexcor.l"
"spritexsize.l"
"spriteycor.l"
"spriteysize.l"
"squeeze.l"
"star.l"
"swap'str.l"
"swap.l"
"translate.l"
"turtle'size.l"
"turtlestate.l"
"uprc.l"
"val.l"
"xcor.l"
"ycor.l"
```

GRAPHIC PRIMER

```
"boot c64 comal"
"c64 comal 0.14"
"comalerrors"
"hi"
"----programs----"
"menu"
"demo 2.1"
"demo 2.2"
"demo 2.3"
"demo 2.4"
"demo 3.1"
"demo 3.2"
"demo 3.3"
"demo 3.4"
"demo 4.1.a"
"demo 4.1.b"
"demo 4.2"
"demo 4.3"
"demo 4.4"
"flurry"
"read'sprite.demo"
"sprite'designer"
"--book listings-"
"demo 2.1.l"
"demo 2.2.l"
"demo 2.3.l"
"demo 2.4.l"
"demo 3.1.l"
"demo 3.2.l"
"demo 3.3.l"
"demo 3.4.l"
"demo 4.1.a.l"
"demo 4.1.b.l"
"demo 4.2.l"
"demo 4.3.l"
"demo 4.4.l"
"---procedures---"
"circle1.l"
"circle2.l"
"circle3.l"
"getbackground.l"
"getborder.l"
"getpen.l"
"getpencolor.l"
"getspritecolor.l"
"getturtlesize.l"
"graphicstate.l"
"heading.l"
"hidescreen.l"
"polygon.l"
"read'sprite.proc"
"showscreen.l"
"showsprite.l"
"spritestate.l"
"spritexcor.l"
"spritexsize.l"
"spriteycor.l"
"spriteysize.l"
"turtlestate.l"
"xcor.l"
"ycor.l"
"--sprite files--"
"flake.dat"
```

UTILITIES

```
"boot c64 comal"
"c64 comal 0.14"
"hi"
"comalerrors"
"----------------"
"generate errfile"
"ram'errors"
">---printer----<"
"dump'1525"
"dump'big'epson.l"
"dump'nec8023a"
"dump'prowriter"
"dumpscreen/demo"
"file'to'print"
"8023p'options4"
">---sprites----<"
"abc.sprite"
"sprite'editor"
">----sound-----<"
"beeper.l"
"fanfare"
"music/demo"
">--disk access-<"
"auto'directory"
"copy&label"
"dir'lister"
"dir'manipulator"
"disk'get/demo"
"disk'protector"
"file'to'screen"
"two'drive'copier"
"two'drive'instru"
"utilities"
"view'directory"
">----modem-----<"
"terminal"
"vt-52.v4"
">----other-----<"
"basic'to'comal"
"color'combo"
"dec'to'hex/demo"
"dynam-data-8"
"find'string/fast"
"find'string/full"
"formatter2"
"hex'to'dec/demo"
"load'screen2.l"
"pie'chart'maker"
"print'seq'files"
"quicksort'number"
"quicksort'string"
"remove comments"
"save'screen.l"
"turtle/demo"
">the following <"
">program is in <"
">basic         <"
"1541backup(free)"
```

SLIDE SHOW 1

```
"slide show"
" directory"
"moire1.hrg"
"moire2.hrg"
"pattern2.hrg"
"nude1.hrg"
"girl.hrg"
"nude2.hrg"
"spacecraft.hrg"
"earring.hrg"
"pattern1.hrg"
"linefig1.hrg"
"caveman.hrg"
"man.hrg"
"tetrad.hrg"
"snail.hrg"
"pattern3.hrg"
"petroom.hrg"
"linefig2.hrg"
"linefig4.hrg"
"linefig3.hrg"
```

SLIDE SHOW 2

```
"slide show"
" directory"
"spiral.1.hrg"
"sue.hrg"
"sesame.st.hrg"
"snoopy.hrg"
"albert.hrg"
"dollar.hrg"
"dip.hrg"
"donald.duck.hrg"
"des.1.hrg"
"7-3hill.hrg"
"music.hrg"
"map.hrg"
"diane.hrg"
"willy.hrg"
"raccoon.hrg"
"sincos1.hrg"
"watch.hrg"
"winston.hrg"
"micrometer.hrg"
```

AUTO RUN DEMO

```
"boot c64 comal"
"c64 comal 0.14"
"comalerrors"
"hi"
">---programs---<"
"menu"
"Arabesque"
"Business Graph"
"Chris's Star"
"Clock"
"Clown"
"Color Swirl"
"Curves"
"Draw Diamond"
"Draw House"
"Example Song"
"Graph Equations"
"Graph Waveforms"
"Graphics Tutor"
"Gutenberg"
"Hilbert"
"Ink Blot"
"Keno"
"Optical Hexagon"
"Optical Illusion"
"Optical Triangle"
"Pie Chart"
"Polyspirals"
"Random Music"
"Slide Show"
"Spirolateral"
"Takeoff Demo"
"Towers of Hanoi"
"----------------"
">  data files  <"
">    follow    <"
">--------------<"
"abc.sprite"
"alpha2.dat"
"blither.hrg"
"calculator.hrg"
"glady.hrg"
"griffin.hrg"
"rocket.sprite"
```

TUTORIAL DISK

```
"boot c64 comal"
"c64 comal 0.14"
"comalerrors"
"hi"
"menu"
"lesson one"
"lesson two"
"lesson three"
"lesson four"
"lesson five"
"lesson six"
"lesson seven"
"lesson eight"
"lesson nine"
"lesson ten"
"lesson eleven"
"lesson twelve"
"lesson thirteen"
"lesson fourteen"
"lesson fifteen"
"lesson sixteen"
"lesson seventeen"
"lesson eighteen"
"lesson nineteen"
"lesson twenty"
"graphics'tutor"
"turtle'tutor"
"numberfile"
"studentfile"
```

SAMPLER DISK

```
"boot c64 comal"
"c64 comal 0.14"
"7777777777777777"
",""""""""""""""""";"
"6error messages!"
```

```
"6    file      !"
"<88888888888888>"
"comalerrors"
","""""""""""""""";"
"6file generator!"
"<88888888888888>"
"generate errfile"
","""""""""""""""";"
"6auto boot prog!"
"<88888888888888>"
"hi"
","""""""""""""""";"
"6seq data files!"
"<88888888888888>"
"information84mar"
"help-comal"
"help-graphics"
"help-sprites"
","""""""""""""""";"
"6comal programs!"
"<88888888888888>"
"see'information"
"see'instructions"
"logo'book'sample"
"snowflake"
"sprite/turtle"
"squiral"
"music"
"bounce"
"sprite'designer2"
"lander"
"create'lander"
"sky'falling"
"create'sky"
"read'directory"
"print'directory"
"expression"
"utilities"
"recursions"
"formatter2"
"file'to'print"
"file'to'screen"
"disk commands"
"c64 comal info"
"remove comments"
"see'roll/demo"
"see'page/demo"
"cursor/demo"
"value/demo"
"shift/demo"
"jiffy/demo"
"quicksort/demo"
"joystick/demo"
"paddle/demo"
"disk'get/demo"
"logical'ops/demo"
","""""""""""""""";"
"6 sprite data  !"
"6   files      !"
"<88888888888888>"
"lander'sprites"
"sky'sprites"
","""""""""""""""";"
"6end of comal  !"
"6demo programs !"
"6              !"
"6the following !"
"6two programs  !"
"6are written in!"
"6   basic      !"
"6   -----      !"
"6do not load   !"
"6them into     !"
"6comal.        !"
"<88888888888888>"
"1541backup(free)"
"single file copy"
```

CARTRIDGE DEMO 1

```
"presentation"
"--copy programs-"
"singledrive'copy"
"dual'drive'copy"
"-external procs-"
"ext.add"
"ext.edit 40"
"ext.sub"
"----package-----"
"pkg.francais"
"-handbook prg's-"
"abs"
"and"
"append"
"at"
"atn"
"bitand"
"bitor"
"bitxor"
"case"
"chain"
"chr$"
"close"
"closed"
"con"
"copy"
"cos"
"cursor"
"data"
"delete"
"dim1"
"dim2"
"dim3"
"div"
"do"
"elif"
"else"
"end"
"endcase"
"endfor"
"endfunc"
"endif"
"endloop"
"endproc"
"endtrap"
"endwhile"
"eod"
"eof"
"err"
"errfile"
"errtext"
"esc"
"exec"
"exit"
"exp"
"external"
"false"
"file"
"find"
"for"
"func"
"get$"
"handler"
"if"
"import"
"in"
"input1"
"input2"
"input3"
"int"
"interrupt"
"key$"
"label"
"len"
"let"
"log"
"loop"
"mod"
"mount"
"not"
"null"
"of"
"open"
"or"
"ord"
"otherwise"
"page"
"pass"
"peek1"
"peek2"
"poke1"
"print"
"printfile"
"proc"
"random"
"randomize"
"read1"
"read2"
"read3"
"read4"
"ref"
"rename"
"repeat"
"report"
"restore"
"return"
"rnd"
"select"
"sgn"
"sin"
"spc$"
"sqr"
"status$"
"step"
"stop"
"str$"
"tab"
"tan"
"then"
"time"
"to"
"trap"
"true"
"until"
"using1"
"using2"
"val"
"when"
"while"
"write1"
"write2"
"write3"
"zone"
"---data files---"
"dat.account'list"
"dat.random'input"
"dat.random'read"
"dat.test'loop"
"dat.visitor"
"dat.visitor'read"
"dat.winners"
```

CARTRIDGE DEMO 2

```
"all'at'once"
"-demo programs--"
"april'fool"
"arabesque2"
"bach'music"
"batch'copier"
"breakout"
"change'unit'#"
"create'fonts"
"curve1"
"curve2"
"differentiation"
"dragon"
"drawing-3d"
"error trapping"
"formatted'list"
"get/set'screen"
"graph1"
"graph2"
"graph3"
"handler1"
"handler2"
"handler3"
"handler4"
"handler5"
"handler6"
"hilbert curves"
"make'eprom'file"
"moire"
"moving'comal"
"moving'frank"
"paddles"
"playscore"
"queens"
"santa'game"
"spiral'circles"
"spirolateral"
"tower'of'hanoi"
"---batch file---"
"bat.commands"
"---data files---"
"dat.bwv779"
"dat.bwv781"
"dat.bwv794"
"dat.screens"
"lst.getnum"
"---font files---"
"font.mirror"
"font.standard"
"---functions----"
"func.binary$"
"func.hex$"
"--shape files---"
"shap.'a'"
"shap.'c'"
"shap.'f'"
"shap.'k'"
"shap.'l'"
"shap.'m'"
"shap.'n'"
"shap.'o'"
"shap.'r'"
"shap.harry00"
"shap.harry01"
"shap.harry02"
"shap.harry03"
"shap.harry04"
"shap.harry05"
"shap.harry06"
"shap.harry07"
"shap.harry08"
"shap.harry09"
```

"shap.harry10"
"shap.harry11"
"shap.queen"
"shap.santa0"
"shap.santa1"
"shap.santa2"

CARTRIDGE DEMO 3

"all'at'once2"
"--demo programs-"
"1520 plotter"
"arabesque2"
"arabesque3"
"batchfile'editor"
"binary'counter"
"check'cartridge"
"curve3"
"curve4"
"extend'color"
"file'card'maker"
"graph4"
"graph5"
"graph6"
"koala'to'2.0"
"picture'loader"
"playscore2"
"protect64"
"read'directory"
"running'men"
"show'character"
"showlibs"
"sidmonitor"
"sound'envelope"
"sprite'editor"
"stampsprite"
"view'fonts"
"--batch files---"
"bat.commands"
"bat.font'cmds"
"---data files---"
"dat.bwv783"
"dat.bwv786"
"dat.bwv801"
"---font files---"
"font.computer"
"font.d&d"
"font.greek"
"font.hebrew"
"font.rooski"
"font.standard"
"----pictures----"
"hrg.northwest"
"hrg.world'map"
"--procs & funcs-"
"func.modem'get"
"proc.plotter"
"proc.show'sprite"
"---shape files--"
"shap.bat0"
"shap.bat1"
"shap.bat2"
"shap.men0"
"shap.men1"
"shap.men2"

CARTRIDGE DEMO 4

"run'me'first"
"--demo programs-"
"all'at'once3"
"another'moire"
"auto'directory"
"bounce"
"cbm'to'comal"
"checkerboard"
"cones"
"copy'seq'file"
"crazy'quilt"
"datacollision"
"draw'flowers"
"draw'hourglass"
"draw'house"
"dual'drive'copy2"
"eight'boxes"
"factorial"
"ink'blot"
"logo'sampler"
"moving'boxes"
"moving'flag"
"mps801'dump"
"music'from'0.14"
"music'player"
"paint'circles"
"pascals'triangle"
"primes"
"print'time"
"quicksort"
"sierpinski"
"son'of'moire"
"spiral'squares"
"spiral'stars"
"starwatch"
"three'pictures"
"---data files---"
"dat.bwv779"
"dat.bwv781"
"dat.bwv783"
"dat.bwv786"
"dat.bwv794"
"dat.bwv801"
"dat.instructions"
"--procs & funcs-"
"func.binary$"
"proc.directory"
"proc.ellipse"
"proc.memorymap"
"proc.quicksort"
"proc.repeat'key"
"proc.stretch'x"
"proc.stretch'y"

COMAL USER GROUPS

If you are just starting with COMAL, one of the best moves you can make is contacting your local COMAL group (see the listing of groups in COMAL TODAY #4). If there is none in your area, consider starting one. There are alot of User Groups available around the US and Canada. Probably over half of the Commodore 64 User Groups now have a **S**pecial **I**nterest **G**roup for COMAL (COMAL SIG). We heard that there are several Commodore Users Groups that have 100 percent of its members interested in COMAL - and one converted into a COMAL Users Group. We really want to publish a complete list of these. Last issue we included a post card for each User Group to fill in and return to us. We are using these to create our list. We hope to include the post card in this isue as well. Make sure your group is represented in our card file (you don't have to be the president of he group to send in a card). We ran out of room this issue, but next issue we hope to print the complete list (it is nine columns long already).

If you wish to **START a group** - send in the card and we will announce it for you (you may be surprised how many other COMAL users are in your area waiting to hear about you). And to get you off to a good start, we are offering a special for new (and old) users groups and schools:

19 different COMAL 0.14 disks (nearly 1000 COMAL programs)
ALL for only $94.05

And you may have noticed that the User Group card asks for a bit of extra information, such as how many members are using COMAL and favorite programs and books. We are happy to report on what alot of the cards had to say:

Indicating COMAL's growth, most groups indicated that the number of members who will have COMAL by January 1985 over NOW (which was Sept/Oct 84) would be about DOUBLE. Typical were 15% now, 25% in Jan; 25% now, 50% in Jan; 20% now, 40% in Jan; 10% now, 50% in Jan; 50% now, 70% in Jan; 60% now, 80% in Jan.

Favorite programs included: Gutenberg (top choice by far), Captain COMAL Gets Organized set, Clue, Screendumps, Pie Chart Maker, Utilities, Tutorial Disk, Directory Manipulator, and a few that said ALL!

Favorite books included: COMAL HANDBOOK (top choice), COMAL FROM A TO Z, and BEGINNING COMAL.

We plan to put out a BEST OF COMAL 0.14 disk in February. This is your last chance to let us know what programs **YOU** consider the BEST. Send us a postcard with your choices!

NEC DOUBLE SIZE GRAPHICS DUMP

by Peter Foiles

Ooops! Peter Foiles informs us that the NEC screen dump routines were not on TODAY DISK #4 as we thought. However, he then kindly sent us an improved version which gives a much larger printout. The source code and object code (for obj'load) are on TODAY DISK #5 as files: BIGDUMP'NEC.OBJ and BIGDUMP'NEC.SRC. Colin Thompson informs me that to call the routine use SYS 52224 for regular size or SYS 51712 for enlarged size. He also warns that the routine doesn't reset the printer, assuming your program will handle it if needed.

COMAL TODAY SUBSCRIBER SPECIALS

•

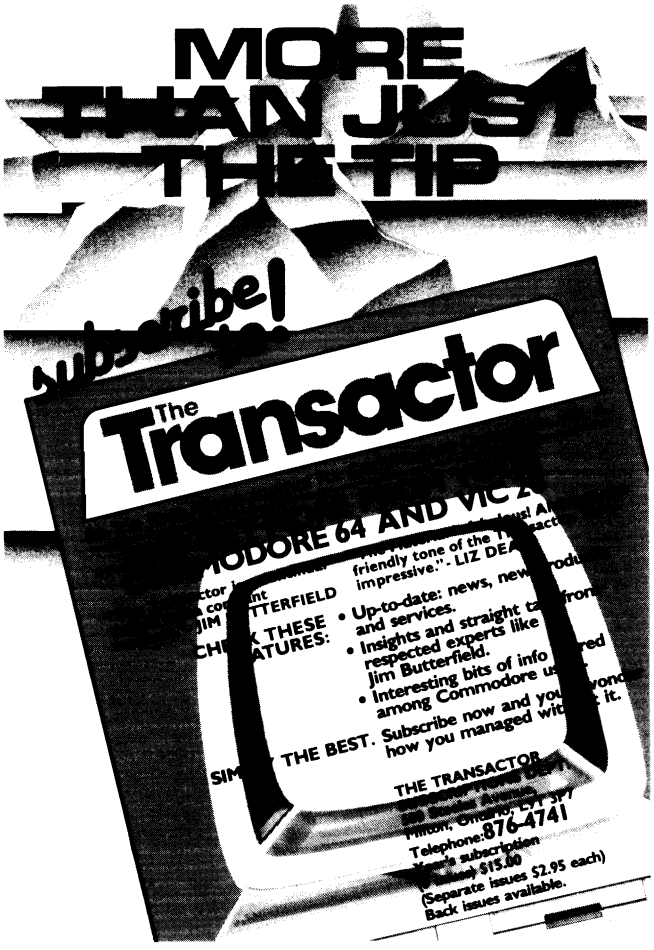
* TUTORIAL DISK - $10 (Reg. $19.95)
* AUTO RUN DEMO DISK - $10 (Reg. $19.95)
* Utilities Disk #1 - $10 (Reg. $19.95)

* Matching disk to the following books: at $4.95 each (Reg. $19.95)
  * Beginning COMAL
  * Structured Programming With COMAL
  * Foundations In Computer Studies With COMAL
  * COMAL Handbook

Prices are good only for new or current COMAL TODAY subscribers. You must clearly indicate on your order that you are a subscriber or regular prices will apply.

COMAL 2.0 CARTRIDGE NOTICE

COMAL Cartridges are not available from Commodore USA. Order direct from COMAL Users Group, USA, Ltd. About 900 have already been shipped. We have about 100 in stock as of December 21, 1984. We may run out before more can be made. If so, prepaid orders will be accepted and placed on a list for first shipment as soon as the cartridges are back in stock.



COMAL SYSTEMS:

Deluxe COMAL 2.0 Cartridge Package (includes manuals and disks) - $128.90
Plain Cartridge (no manuals, no disks - meant for user groups and schools) - $99.95
COMAL 0.14 Starter Kit, $29.95 (Three different disks and book COMAL FROM A TO Z)
PET/CBM Disk Loaded COMAL 0.14 on 4040 disk with programs, $14.95
CBM 8096 COMAL version 2.00 disk (includes COMAL HANDBOOK), $149.95
(still awaiting final version - advance orders accepted)

SUBSCRIPTIONS, NEWSLETTER AND DISK:

COMAL TODAY newsletter subscription - $14.95 (specify starting issue #)(Sample $2)
=> (Canada add $5, outside North America add $30 for special handling)
TODAY DISK Subscription (6 disks) - $59.90 (specify starting disk number)

BOOKS AND BOOK/DISK SETS:

>CARTRIDGE SUPPORT PACKAGE (2 books, 2 disks - included with Deluxe Package) - $65
=>**COMAL HANDBOOK, 2nd Ed, 470 pgs, Len Lindsay - $18.95 (included with Deluxe Cart)**
optional matching disk - $19.95 (only $4.95 to COMAL TODAY subscribers)
CAPTAIN COMAL GETS ORGANIZED, second printing, (book and disk) - $19.95
=>BEGINNING COMAL, 333 pages, Borge Christensen - $22.95 (**new price**)
Second Printing due back in stock direct from England early 1985
optional matching disk - $19.95 (only $4.95 to COMAL TODAY subscribers)
STRUCTURED PROGRAMMING WITH COMAL, 266 pages, Roy Atherton - $26.95 (**new price**)
optional matching disk - $19.95 (only $4.95 to COMAL TODAY subscribers)
FOUNDATIONS IN COMPUTER STUDIES WITH COMAL, 313 pages, John Kelly - $19.95
optional matching disk - $19.95 (only $4.95 to COMAL TODAY subscribers)
COMAL FROM A TO Z, third printing, Borge Christensen - $6.95
=>CARTRIDGE GRAPHICS AND SOUND - $9.95 (included in Deluxe Cartridge Package)
COMMODORE 64 GRAPHICS WITH COMAL, Len Lindsay (due January 1985)
STARTING WITH COMAL, Ingvar Gratte (delayed about 1 year by publisher)
CAPTAIN COMAL'S FIRST WORKBOOK, Gordon Shigley - $6.95
=>THE COMAL LIBRARY OF FUNCTIONS AND PROCEDURES, Kevin Quiggle (book & disk) - $19.95
=>CAPTAIN COMAL'S GRAPHICS PRIMER, Mindy Skelton (book & disk) - $19.95

DISKS:

=>Cartridge DEMO Disks, $19.95 each: #1, #2, #3, #4 (#1 & #2 included with Deluxe Cart)
TUTORIAL DISK, $19.95 (only $10 to COMAL TODAY subscribers)
AUTO RUN DEMO DISK, $19.95 (only $10 to COMAL TODAY subscribers)
COMAL SAMPLER DISK, $19.95 (the original C64 COMAL disk)
=>SLIDE SHOW Disks, $10 each: #1, #2
TODAY DISKS, $14.95 each: #1, #2, #3, #4, #5 (6 disk subscription $59.90)
USER GROUP DISKS, $10 each: #1, #2, #3, #4, #5, #6, #7
UTILITIES DISK #1, $19.95 (only $10 to COMAL TODAY subscribers)

OTHER:

=>CHEATSHEET keyboard overlay for COMAL 0.14 - $3.95 (plus $1 each handling)
=>McPen Light Pen (with COMAL 2.0 sample programs) - $49.95

Note: ALL orders add minimum $2.00 shipping/handling. Add $2 extra each book after 1st.
North America orders only. All orders prepaid US funds on US Bank. No COD.
Company and School Purchase Orders accepted only if accompanied by payment in full.

ORDER LINE: 1-800-356-5324 ext 1307 (VISA & MasterCard Orders Only - no specials!)
Questions / information call our INFO LINE: 608-222-4432 from 9am to 4pm or write:
COMAL Users Group, U.S.A., Limited, 5501 Groveland Ter, Madison, WI 53716-3251